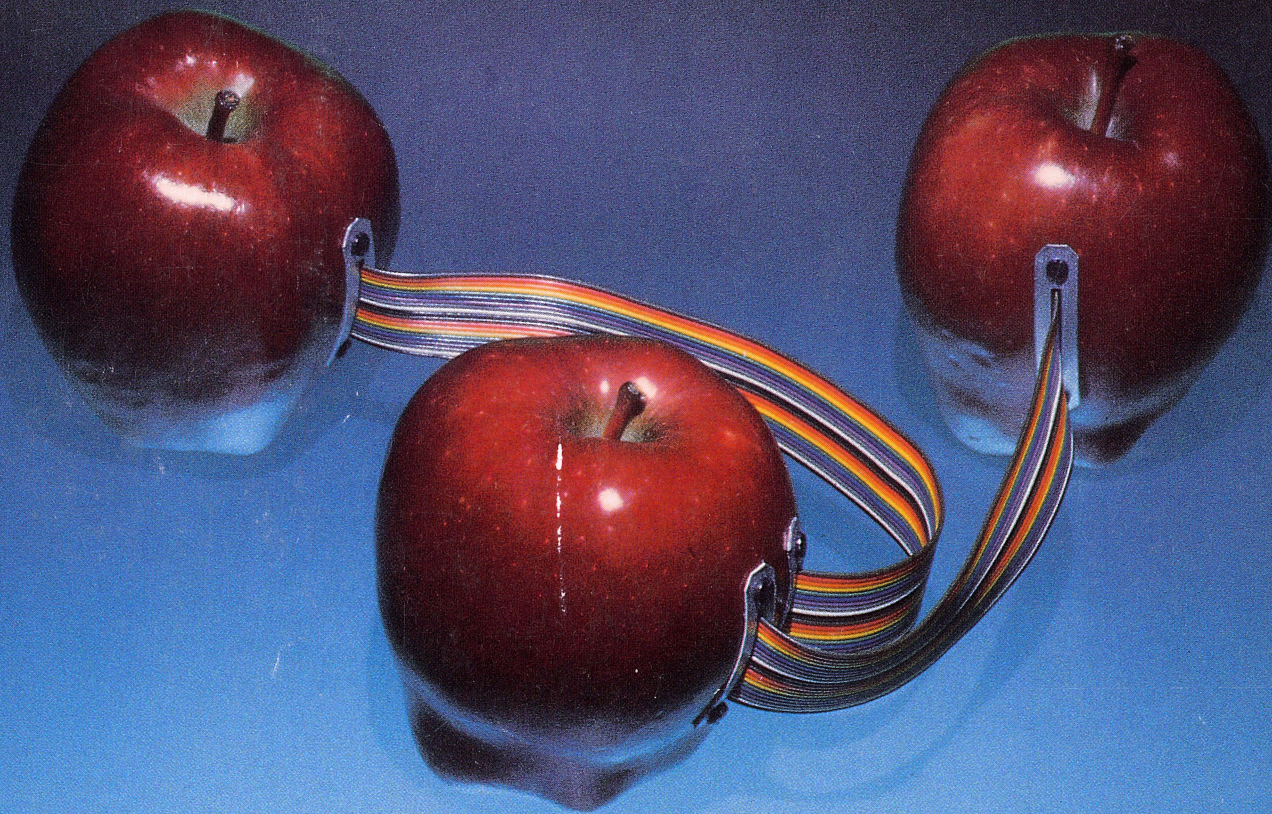


1717

INTERFACING & DIGITAL EXPERIMENTS WITH YOUR APPLE®



BY CHARLES J. ENGELSHER

**INTERFACING &
DIGITAL EXPERIMENTS
WITH YOUR
APPLE®**

INTERFACING & DIGITAL EXPERIMENTS

WITH YOUR

APPLE®

BY CHARLES L. ENGELSHER



TAB BOOKS Inc.

BLUE RIDGE SUMMIT, PA. 17214

Apple is a registered trademark of Apple Computer, Inc.

FIRST EDITION

FIRST PRINTING

Copyright © 1984 by TAB BOOKS Inc.

Printed in the United States of America

Reproduction or publication of the content in any manner, without express permission of the publisher, is prohibited. No liability is assumed with respect to the use of the information herein.

Library of Congress Cataloging in Publication Data

Engelscher, Charles J.
Interfacing and digital experiments with your Apple.

Includes index.

1. Computer interfaces. 2. Apple Computer. 3. Digital electronics. 4. Computer input-output equipment.

I. Title.

TK7887.5.E54 1984 621.3819'5835 83-24169
ISBN 0-8306-0717-X
ISBN 0-8306-1717-5 (pbk.)

Contents

Introduction	vii
1 Fundamentals of Digital Electronics	1
The Challenge of Digital Electronics—The Microcomputer Solution—Learning about Digital Devices—Digital Devices as Black Boxes—Discrete Components and Device Characteristics—Families and Subfamilies—Scales of Integration—Binary and Hexadecimal Numbers	
2 The Digital Desktop Laboratory	13
The Game Port Connection—Turning the Apple into a Digital Logic Trainer—The Rest of the Logic Lab—Introduction to the Experiments—Experiment 1, Setup	
3 The Six Basic Logic Functions	37
Combinational SSI Devices—Experiment 2, NOT and IS—Experiment 3, AND—Experiment 4, NAND—Experiment 5, OR—Experiment 6, NOR	
4 Boolean Methods	61
The Rules of Boolean Algebra—Relating Inputs and Outputs—Introduction to the Design Experiments—Experiment 7, SSI Design I—Experiment 8, SSI Design II	
5 Discrete Electronic Components and Laws	91
The Resistor and Associated Laws—Capacitance, Time Constants, and Reactance	
6 The Diode and Transistor	123
Semiconductor Materials—The Diode—The Essentials of Transistor Action—Using the Transistor as a Switch—Experiment 9A, Diode and Transistor Testing—Experiment 9B, Transistor Current Gain and the Overloaded Sink	

7	TTL Internals	161
	The Electrical Black Box—Inside the Black Box—The Key Parameters of TTL—TTL and Bus Oriented Computer Logic—Experiment 10, LSTTL Sink Current Measurement and Augmentation	
8	SSI Sequential Devices	189
	Types of Sequential Devices—R/S Flip-Flops—Clocked Logic Concepts—Experiment 11, R/S Latches—D and T Flip-Flops—Experiment 12, The LS74 Flip-Flop—Experiment 13, the LS75 Quad D-Latch—The J-K Flip-Flop—Experiment 14, The J-K Flip-Flop—Monostable Devices or One Shots—Experiment 15, The One Shot	
9	MSI Sequential Devices	233
	Medium Scale Integration—Ripple Counters—Experiment 16, MSI Ripple Counter—Synchronous Counters—Experiment 17, The Synchronous Up/Down Counter—Shift Registers—Experiment 18, The Shift Register—MSI Latches	
10	Combinational MSI Devices	269
	Code Conversion and Data Routing—Experiments for MSI Combinational Functions—Experiment 19, Priority Encoder—Experiment 20, Binary Decoder/Demultiplexer—Experiment 21, BCD Decoder—Experiment 22, Seven-Segment Decoder/Driver—Three State Devices—Exclusive-OR Functions	
11	An Introduction to Interfacing	299
	The I/O Triad—The System and I/O Memory Maps—I/O Address Decoding—On-Board I/O Circuitry—Simple Applications Circuits—LORES/HIRES Joystick Project—Closing Comments	
	Index	338

Introduction

This book began with a simple idea: to teach the basics of digital electronics in a hands-on manner using the desktop microcomputer as the primary learning tool.

This primer is not a cookbook of circuits. Instead, it provides an approach that will allow you to acquire a basic but thorough knowledge of digital devices and techniques and the ability to apply these skills to more complex and interesting projects.

Because it is assumed that you have little or no knowledge of electronics (digital or otherwise), one of the major goals of this book is to provide you with a knowledge of essential electronics. Too many books assume this knowledge and jump into digital electronics right away. The other broader and more intensive emphasis is on the digital IC chips themselves: what they do, how to specify them, and how to use them to perform simple tasks. Familiarity with basic electronics and with a variety of IC chips from the major functional categories will together give you a firm foundation for future progress. Some additional, specific aims include helping you

gain competence in reading and using the technical data and applications literature, in designing small-scale circuits for specific functions, in interpreting digital schematics (such as the Apple main board diagram), and even interfacing your computer for simple I/O (input/output) applications.

Once you've been grounded in these fundamentals everything else—including computer hardware applications—are nothing more than extensions to this knowledge. The intention is to give you a firm base for advancement.

TOOLS

Learning demands doing. Digital electronics is as much a set of skills as it is a body of knowledge. Your competence in this field depends as much on your working with a variety of digital devices as it does on knowing their function. Handling, testing, and building a circuit using a device gives you a range of stimuli not obtainable from passive study. The more ways you are exposed to a subject, through the fingers as well as through the brain, the better you will understand it.

Like many micro owners who want to learn about the digital hardware, you may be hesitant and uncertain about how to satisfy your curiosity. The would-be experimenter faces the prospect of first acquiring a logic trainer, boxes of parts, and barrels of equipment before embarking on his projects or studies. This costly and time-consuming process is unnecessary. A powerful educational tool is already at your disposal—the Apple computer itself!

The microcomputer has been touted for a long time as a “solution in search of a problem” and as a “universal tool.” So it seemed logical that this powerful all-purpose machine could be adapted to teaching the fundamentals of digital hardware.

By exploiting the economy and simplicity of the Apple game port, you can turn this microcomputer into a digital desktop laboratory with very little effort and minimal expense. You will control game port signals via an Applesoft utility program—the Breadboard-In Software—that converts the monitor screen into a dynamic, reconfigurable display. What happens in the circuit is reflected on the screen, from moment to moment, even as you change signals. This arrangement eliminates most of the inconvenient preliminaries, yet retains all the advantages of the hands-on experimental approach.

METHOD

Because a broad range of material is covered in the Apple Digital Hardware Primer, particular attention is given to the method of presentation. The value of maintaining a consistent level of difficulty, while at the same time building on preceding material, is obvious.

I don't cover every imaginable topic. Rather, the coverage is selectively deep in what you have to know. The emphasis is on basic skills and key concepts.

In this book you will do more than just demonstrate the functions of many digital IC chips. You learn how to describe their function using formal methods and to use them in design by means of Boolean algebra. Also, you learn not only what they do, but how they work—the electronics inside the IC packages. This allows you to use them safely and effectively in actual physical circuits. Further, the

principles and practices introduced are related to computer operation—bus organization, interfacing principles, memory decoding, etc.

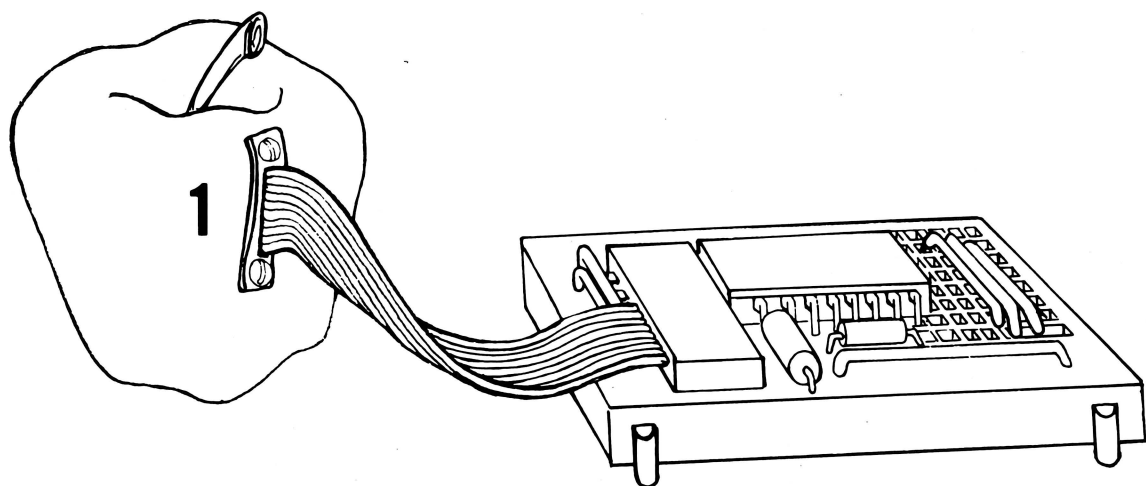
After a general survey in Chapter 1, you are shown how to set up your desktop digital laboratory in Chapter 2. Other “tools” are developed as part of the overall organization. You will first study the digital IC as a “black box,” as a functional unit rather than as an electronic circuit. Function is emphasized. You learn about basic “building block” chips (combinational SSI devices) in Chapter 3, and then see how formal Boolean methods can be applied to digital design using this important group of ICs (Chapter 4).

Chapters 5, 6, and 7 comprise the “essential electronics” section. You learn about discrete electronic components and their laws of operation (capacitors, resistors, diodes, and transistors). Then you apply this knowledge to the operation of a typical digital IC. At the end of this section, you will be quite comfortable with concepts like fan-out, despiing, current sinking logic, capacitive loading, and totem-poles. Data manuals and the technical specifications they contain will no longer be a mystery. You'll use them with facility and confidence.

You then can tackle the area of sequential devices in Chapters 8 and 9. Both SSI and MSI levels of integration are detailed, with representative chips from each category. Then in Chapter 10, the important MSI combinational chips are covered.

In all of the above chapters, experimentation plays a critical role in the presentation. Sometimes problems are presented as part of the experiments, so that they are more than mere demonstrations.

Chapter 11 concludes the book with coverage of some important topics in computer hardware applications under the general heading of interfacing. This chapter rests on the foundations laid down in the previous material. General principles are supported by a chip-by-chip description of key I/O circuitry on the Apple main board. Memory mapping and address decoding are dealt with in detail, using the Apple as an example. A few projects using resistive transducers for simple A/D conversion close the book.



Fundamentals of Digital Electronics

The digital integrated circuit is more than just a popular symbol for modern electronic technology. It is the basis for learning about that technology. In fact, the rationale for this book is that in-depth knowledge of representative digital integrated circuits—how they work and what they do—is the basis for your competence in digital electronics. In this chapter, you'll survey the role of the digital IC and get a preview of the material to be covered in later chapters.

THE CHALLENGE OF DIGITAL ELECTRONICS

Digital integrated circuits are more than just a new group of electronic components. A phenomenal amount of theoretical knowledge and engineering savvy has gone into the creation of these electronic marvels, popularly known as IC chips. Microelectronics, the technology behind these devices, has created entirely new areas of activity and produced dramatic changes in existing ones.

Consumer electronics, including home entertainment and personal computing, global satellite

communications, office and factory automation, medical diagnostics and new graphics design tools are just a few examples of these new applications. Certainly, the IC chip deserves to be the symbol of this electronics revolution. Its impact on daily life is obvious even to the proverbial man on the street.

For you, the microcomputer owner, digital integrated circuits hold more than a mere casual interest. Your machine would not exist without them. More than that, you may have sensed that a knowledge of digital electronics opens up new possibilities—as a hobby, as an enhancement to your present job or as the basis for a new career. Whatever your specific motives, practical applications or sheer curiosity, it is assumed that you do have a strong interest in learning about digital hardware. It's also assumed that you possess little if any electronics knowledge, but that you are willing to put in the extra effort to learn the basics.

Digital electronics is a complex field. Even the more elementary material is challenging. There are so many terms, principles, and concepts that the

subject begins to take on the characteristics of a language. Thorough grounding in the basic concepts is especially necessary in digital electronics in order to avoid superficial cookbook knowledge. The whole concern of the first half of this book is to help you gain mastery of the fundamentals.

However, the major problem is not the difficulty of the subject matter, because your willingness to tackle it is taken for granted. Rather, the problem is that of paraphernalia. Let me explain.

Paraphernalia is a long story with a sometimes unhappy ending. It begins with the question, "What parts, tools, materials and equipment do I need to get started in learning digital electronics?" It continues with searches through the catalogs and the making of long lists. After you shop, then you order; then you wait, often for quite a while. When your digital laboratory is assembled you can rightfully feel proud. And also a lot poorer. It is usually only when you start paying the bill that you realize that perhaps you overbought. You didn't *really* need that fancy dual trace oscilloscope, that function generator or that elaborate logic trainer-prototyping apparatus. In addition, it may take a while to gain proficiency in the use of this equipment before you can even begin experimenting.

Quite frankly, hardware enthusiasts do spend a lot of time and money on equipment and materials. There seems no other way to acquire the necessary hands-on knowledge of digital electronics other than to follow the scenario suggested above. And of course, quality equipment is worth it in the long run if you actually need it. But is it necessary to spend all that money as a beginner? Isn't there a better way?

THE MICROCOMPUTER SOLUTION

A powerful, all purpose tool already in your possession will help you to solve this hardware paraphernalia problem: the Apple microcomputer.

By exploiting the economy and simplicity of the Apple game port, you can turn this machine into a digital logic trainer/breadboard system with minimal effort and expense. You control the game port signals and monitor them via an Applesoft utility program—the Breadboard-In-Software

(BDIS)—which turns the monitor screen into a dynamic, reconfigurable tabular display. All the signals sent to and received from the circuit under study are displayed in an easy to read form. This arrangement eliminates all of the costly preliminaries, yet provides all of the advantages of a conventional hands-on system.

This computer-based trainer offers other advantages. Meaningful labels can be assigned to the signals of the device or circuit under examination. Pin-outs may be specified for ease of circuit hookup. Display format may be altered for convenient viewing. A printout can be obtained for permanent records of your experiments if you desire. Also, BDIS continually monitors the outputs of the circuit under study and automatically updates the display.

All that is required is a jumper cable, solderless breadboard strip, and a half-hour or so to type in the Breadboard-In-Software (BDIS) utility program.

At this point it is tempting to present the BDIS utility, show how to make the simple game port connection, and then procede to some "flashing lights" demonstrations. We'll do these and many other things in the next few chapters. But first it is necessary that you have a clear idea of how you are going to study digital devices and circuits.

LEARNING ABOUT DIGITAL DEVICES

You can look at any specific digital IC device from several different standpoints:

- ☐ As a sealed black box which performs certain logical functions.
- ☐ As a collection of discrete electronic components integrated on a tiny chip of silicon.
- ☐ As the member of a particular family of devices which share the same electrical characteristics.
- ☐ As representing a particular level of component density per unit area of silicon—a level or scale of integration.

Virtually all the things you should know about digital ICs fall into one or the other of these

categories. The first category mentioned, the black box view, stands apart from the rest. The next two categories both relate to what goes on inside the IC. The last category refers to the levels of functional complexity of ICs. Let's look at each item in turn.

DIGITAL DEVICES AS BLACK BOXES

The *black box* is a convenient engineering concept which is invoked whenever there is a need to simplify a complex process or system. If the object under study appears too complex you may, figuratively speaking, draw a line around it. By boxing in the system, you hide from view a lot of confusing internal detail. This technique makes it easier to understand how outputs are related to inputs. For the moment you don't care about what is going on inside the system, how the electrons do their chores, or how the system was manufactured (Fig. 1-1).

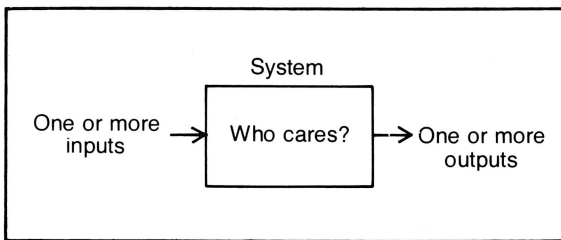


Fig. 1-1. With the black box approach, you don't care about the mechanism inside. Your concern is with function—how the inputs are related to outputs.

Digital integrated circuits are fabricated on a microscopic scale on wafers of silicon, commonly sealed in oblong, black plastic packages, and connected to the outside world by small metal tabs or pins. The resemblance between the black box idea and digital IC's suggested in Fig. 1-2, is no coincidence. They literally are little black boxes and were designed to be used as such.

Entire circuits, consisting of dozens, hundreds or thousands of components have been packed into them according to strict design guidelines. Because complex circuits have been put into standard, easy-to-handle packages, the user is free to concentrate on the main task of circuit design: how to configure individual chips into a system using the

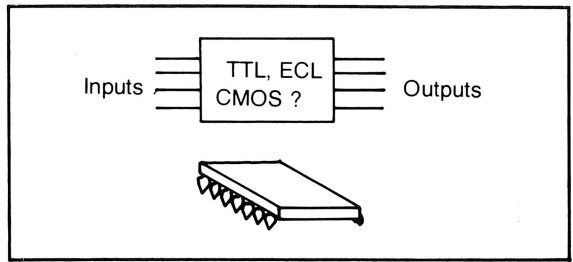


Fig. 1-2. The IC as black box. In learning about the devices from a functional standpoint, the electronics is incidental.

chips as circuit building blocks. Likewise the student can begin learning about devices and simple circuits by studying digital functions. He does not have to worry about device electronics or electrical characteristics at the outset.

This is the approach taken in Chapters 3 and 4, where the emphasis is placed on what the device does and on the methods used to describe its function.

DISCRETE COMPONENTS AND DEVICE CHARACTERISTICS

Naturally, the engineer or student will have to take into account a number of practical matters when he actually starts building circuits. Knowledge of power consumption, allowable signal and power supply voltage levels, and current loading limits is essential. What this really boils down to is some working knowledge of basic electronics, at least enough to understand device specifications. But just how much knowledge is enough?

You need only learn about three limited areas: passive elements, the transistor as a switch, and the operation of a typical digital circuit in terms of these individual components.

Passive elements include resistors, capacitors and diodes; the rules governing these components are straightforward. Transistor operation is not as simple. However, we are only concerned with its operation as a switch, not as an amplifier, and this makes the topic much easier. Last, we will look at how these components are configured on a typical IC chip and talk about the IC's *electrical parameters*.

Three chapters—Chapters 5, 6, and 7—are devoted to this material and include supporting ex-

periments. These chapters may demand more effort from some of you, especially if you've forgotten or never learned such things as Ohm's Law and Kirchhoff's Law. But remember, the goal is not to turn you into an IC design engineer, but just to teach you some basic electronics.

As a result of your efforts you will be able to open up any IC data manual and read it intelligently. Since the technical data and applications manuals from the many chip manufacturers are valuable information sources in their own right, this is an important skill. The discussion of digital IC characteristics in Chapter 7 includes the type of specifications you would normally encounter in data manuals. Therefore, by the time you finish this section, you should be fairly adept at reading and using them.

FAMILIES AND SUBFAMILIES

The subject of families of IC devices is really an extension of the subject just discussed, discrete component electronics. As mentioned, ICs are the result of rigid design criteria which define the electrical properties of the device packages. IC design also involves building in the black box function; the specific logical, arithmetic or other task that the device is to perform. Obviously you may require that a device fulfill a given function and be able to operate in a wide variety of settings. For instance you may want the fastest device currently available and require the lowest power consumption available. Unfortunately you can get either characteristic alone, but not both combined in the same chip. This is known as the speed/power tradeoff. There are other tradeoffs as well.

In other words, whatever property you care to discuss—noise immunity, power use, cost, current output, operating temperature range, speed, etc.—you will have to make compromises. Improvement in one factor is usually realized to the detriment of one or more of the others.

The reason for these real-world constraints involves the type of semiconductor used to make the device as well as the fabrication method employed. The nature of your compromise is very much determined by the *family* of IC device that you

choose. Most devices fall into either one of two broad semiconductor families: those based on the *bipolar transistor* and those based on the *field-effect transistor*. See Fig. 1-3.

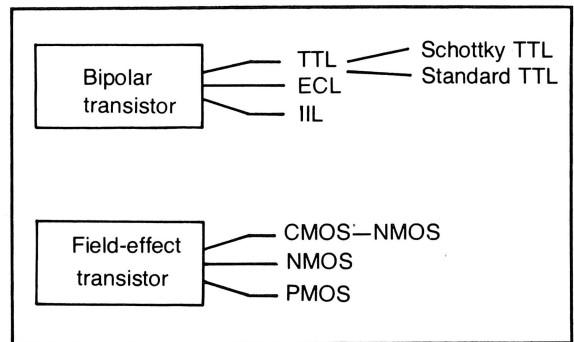


Fig. 1-3. The major families and subfamilies of digital ICs are based on the two main types of transistor. The electronics is important when it comes to actually building the physical circuit.

The bipolar transistor, the original one invented in 1948, is the basis for a major family called *transistor-transistor-logic*, or *TTL* for short. Other bipolar IC families are *ECL* (emitter coupled logic) and *IIL* (integrated injection logic). Each is the result of careful consideration of user requirements and of fabrication techniques. Each has its own strengths and weaknesses. For example, both ECL and TTL are used in digital logic applications. ECL is lightning fast, but also power hungry, and very demanding of precise power supply voltages. TTL is also fast, though not so fast as ECL. However, it uses less power and is less finicky in regard to supply voltage.

Integrated injection logic, the other major bipolar family, is used in linear applications. TTL and ECL are primarily digital devices. Linear IIL devices include oscillators, voltage to frequency converters, modulators, and amplifiers. This family will not be considered further.

The other major family of IC is based on the field effect transistor, which operates on slightly different principles than the bipolar transistor. The various device types in this group are collectively known as *MOS* devices, metal oxide on silicon.

MOS family names refer to the type of semi-

conductor, positive or negative type, used in the given family. In *PMOS*, the charge carriers are positive (holes). In *NMOS*, the charge carriers are negative (electrons). In *CMOS*, (complementary MOS), field effect transistors made from both types of semiconductor are used. CMOS also has high performance cousin HCMOS.

At this stage, the important thing to keep in mind about MOS ICs is that they use very little power in comparison to TTL and are proportionally slower. In particular, the CMOS family (complementary MOS) consumes only a fraction of the power of comparable TTL devices, making CMOS very useful in battery powered applications where high speed is not essential. A second major advantage of CMOS over TTL is that you can safely use a wide range of power supply voltages.

However, all MOS devices have a handling problem: they are inherently sensitive to static discharge. Older CMOS versions required that the user be attached to earth ground via conductive floor mats and wrist straps, and that he observe certain storage and handling precautions when working with these ICs. If you rub your feet on a carpet on a dry winter day and then touch a metal object, you will have a clear idea of how static discharge can destroy a delicate MOS chip. This problem has been partially remedied in more recent designs. However, it is still possible to destroy one of these chips with static electricity if you don't

follow these precautions.

Where does all this leave us in terms of choosing the best family on which to learn?

Well, there is a subfamily of TTL which meets the student's and the experimenter's needs admirably that is called *low power schottky ttl* (LSTTL). LSTTL devices are based on the Schottky design enhancement of the traditional bipolar transistor. Schottky transistors are faster yet conservative of power. This means that LSTTL devices enjoy an excellent combination of both high speed and modest power drain. At the same time, they have an edge over CMOS in terms of ruggedness and are also a bit cheaper than CMOS. Because of these last factors, LSTTL has been chosen for the experiments in this book.

Table 1-1 provides a quick comparison of CMOS and LSTTL, and covers the points just mentioned.

SCALES OF INTEGRATION

Scale of integration refers to the number of components that can be packed onto a unit area of semiconductor. The number of transistors per square millimeter of silicon would be an example. In practice chip density or *scale of integration* relates to the finished IC package as you see it on the shelf. The more components that manufacturers can miniaturize and integrate in a given area, the higher the scale of integration.

Table 1-1. Some Typical Device Characteristics for Two Popular IC Families.

Characteristic	LSTTL	CMOS
Power Supply Voltage Range (in volts)	+4.5 - 5.5	+3.0 - 15
Power Consumption per Gate (in milliwatts mW)	1.0	.01 - .1
Speed or Propagation Delay (in nanoseconds ns, i.e., billionths of a second)	10	100 - 150
Ruggedness (in handling)	excellent	satisfactory *
Cost	low	low - moderate
Noise Immunity	good	very good

* CMOS demands certain handling precautions, is more sensitive to static discharge and to improper input currents.

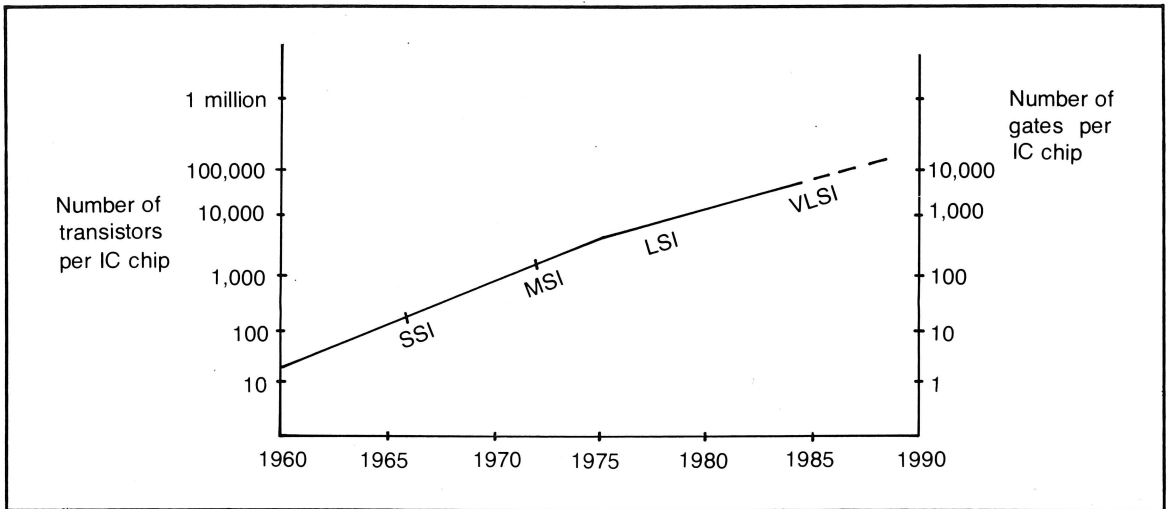


Fig. 1-4. The growth of IC complexity or scale of integration in the past two decades.

Referring to Fig. 1-4 and Fig. 1-5 and to Table 1-2 you can see that *small scale integration* devices (SSI) made their appearance around 1960. They have, by definition, less than 100 transistors per IC device. (The term *device* means the functional unit on the chip: a flip-flop or a gate for instance). The smallest logic element is a circuit composed of transistors, resistors and diodes, often called a *gate*. A typical gate contains from 5 to 9 transistors. The gate count per device is more meaningful than the number of transistors per device. Hence, SSI IC

devices are usually defined as having a maximum of 11 gates.

In the mid to late 1960's, *medium scale integration* (MSI) became available. Digital ICs at this level ranged from 100 to 1000 transistors or 12 to 100 gates per digital device. MSI devices provide many of the workhorse functions of digital circuits: counters, registers, decoders, etc.

Large scale integration (LSI) is defined as having from 100 to 1000 gates, or up to 10,000 transistors per device. More recently, VLSI (*very large*

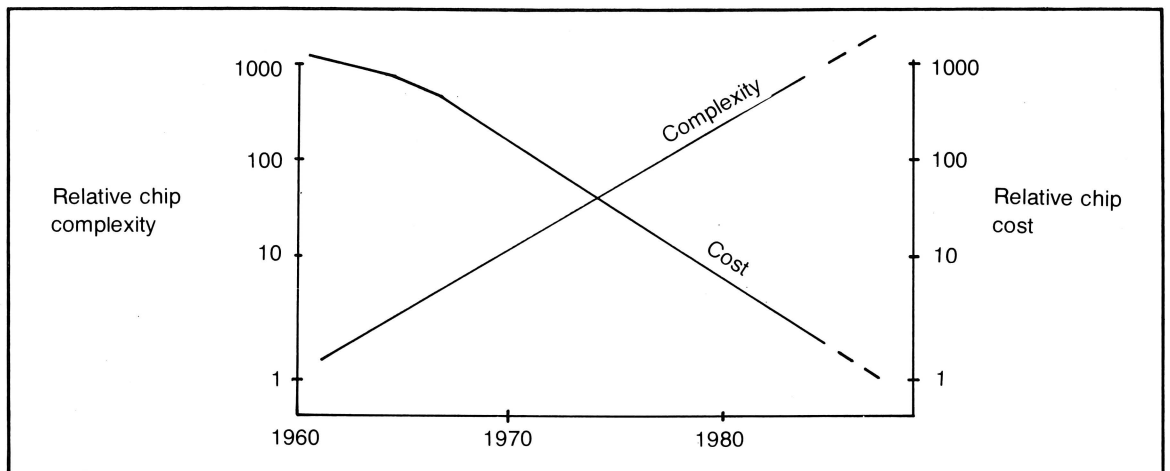


Fig. 1-5. The rise of IC complexity has been accompanied by dramatic decreases in cost.

Table 1-2. Number of Transistors and Gates Per IC Device as a Function of the Scale of Integration.

Scale of Integration	Approximate Time Of Introduction	Transistors Per Chip	Gates Per Chip
VLSI	1980	> 100,000	> 10,000
LSI	1970	1000 - 100,000	100 - 1000
MSI	1965	100 - 1000	12 - 100
SSI	1960	< 100	< 12

scale integration) came into being. VLSI can contain up to 10,000 gates or 100,000 transistors per package. *Ultra Large Scale Integration* is also in the works. It will contain up to 1 million transistors per device. LSI, VLSI and ULSI are exemplified by microprocessor chips and by large (64-bit and 256-bit) computer memory.

The whole point can be summarized in one sentence. **Impressively complex functions, in ridiculously small packages are available at astoundingly low cost.**

As one ascends the scale of integration, one sees more and more complex and sophisticated functions. Cost has dropped off dramatically as the complexity has risen. These changes have been exponential. The cost of computer hardware is decreasing by a factor of about ten every seven years or so. Your two or three thousand dollar desk-top system of today would have carried a price tag in 1960 of close to a million dollars for equivalent functions and with consideration for space, power and maintenance figured in.

Another way to view this phenomenally rapid development from the black box perspective is suggested in Fig. 1-6. In the pre-1960 days, circuits were built largely from individual electronic components. Since then there have been a series of major steps in which the prior technological scale was condensed and integrated into a denser and more complex black box. In this sense, the digital system designer may not care about the MSI level functions that have been packed into the VLSI microprocessor chip that he is working with. He simply uses this chip as a functional black box, along with many other black boxes, in the digital system he is creating.

Despite these advances, the lowly SSI device

still occupies a very important place. As we'll see later, SSI chips serve as a sort of glue which holds the more complex chips together. Occasionally, some simple circuit functions are best realized entirely by SSI chips rather than by higher level devices.

The educational role of SSI is equally as important as their design role. Understanding SSI is straightforward because of its relative simplicity. You can examine the outsides of SSI chips and understand all of the basic digital logic functions. You can also examine their insides and learn the basis for their electrical characteristics and how to use them in practical circuits. Further, once you master SSI, MSI components present no real difficulty, because they are just extensions of the SSI building block functions. Subjects that fall into the LSI range, such as microprocessors, memory, in-

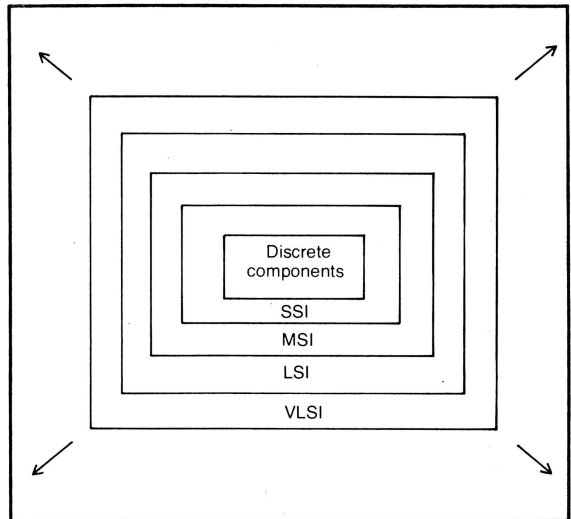


Fig. 1-6. The black box of today is swallowed by the next larger scale of integration of tomorrow.

terfacing and computer hardware applications are then quite accessible.

BINARY AND HEXADECIMAL NUMBERS

You should have some familiarity with binary and hexadecimal number systems at the outset. An ability to translate a number in either system to its decimal equivalent is all that is really required, so the following will be brief.

The simplest and hence most reliable number system is the binary number system. In this system there are only two values for each digit: 0 and 1. Numbers in this system are based on the power of two, rather than on the power of ten, as in the familiar decimal system. Binary is said, therefore, to be a *base-two* system.

Binary is ideally suited to digital circuits for one main reason: lack of ambiguity. In digital circuits, the circuit elements can assume only one of either of two states or values. Such a circuit element can be either on (conducting) or off (nonconducting). This is analogous to a mechanical switch being closed or open. In fact, the basis for all digital circuits is nothing more than an electronic version of the mechanical switch. In short, because there are no intermediate values between high and low, 0 and 1, or on and off, digital systems are by their nature, very, very reliable. (This is unlike analog circuits, which operate on a continuum of values, with the exact state of the circuit element always approximate, always slightly uncertain).

In digital circuits, the binary values of 0 and 1 correspond to two different voltage levels: high and low. These two levels are called *logic high* and *logic low* because they represent the two logical states: true and false. In the LSTTL logic family the nominal values of these two voltage levels are 0 and +5 volts for 0 and 1, respectively. (There is actually a range for valid logic low/0 and logic high/1 voltage levels, something detailed later on.)

Binary Representation

Let's formalize a little. A **bit** is a single digit in the binary number system. It can assume the values of 0 and 1, just as a decimal digit can assume a value in the range of 0 to 9. Just like decimal digits, binary

digits or bits have a *place value*. For instance, in the number 539, the place values are from right to left: units, tens and hundreds. More generally, the place values are 1, 10, 100 and so on, proceeding by a factor of 10 for each place to the left. The leftward most digit is the most significant digit, or MSD, because it has the highest place value or numerical weight.

How this works out for the binary number system is illustrated in Fig. 1-7. A four bit binary number is given in Fig. 1-7A, with the places called d, c, b, and a from most to least significant bit. The positions for these bits are numbered 3, 2, 1, and 0 respectively. Since this is a base-two number system, the place value for each position is given as:

Place Value = 2^x where x is the bit position.

That is, the place values in the binary scheme are, from right to left, 1, 2, 4, 8, and so on by powers of 2.

The number of values represented by a certain number of bits can be figured out quite easily. You know that a three digit decimal number can assume any one of 1000 values when you include 0. This is just 10^3 : you simply raise the number base by the number of digits. In binary, this becomes:

Table 1-3. Equivalence of Binary, Decimal, and Hexadecimal (see text).

Decimal	Binary	Hexadecimal
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10	1 0 1 0	A
11	1 0 1 1	B
12	1 1 0 0	C
13	1 1 0 1	D
14	1 1 1 0	E
15	1 1 1 1	F

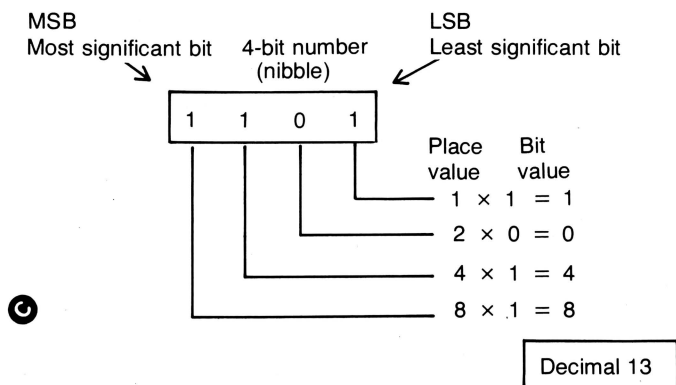
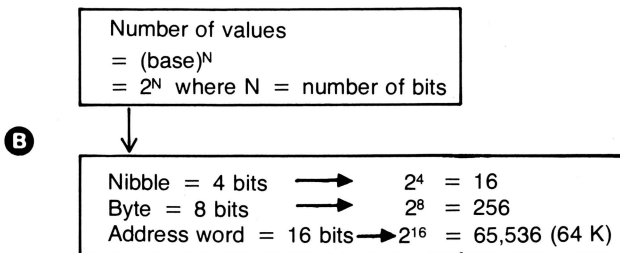
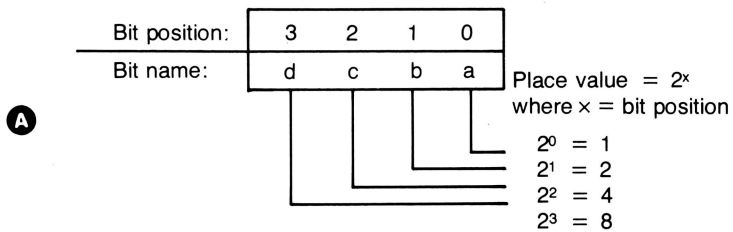


Fig. 1-7. Binary Representation. Refer to the text.

Number of Values = (base)^N where
 N = number of bits
 or
 Number of values = 2^N

This works out to 16 values (0 to 15 decimal) for a four bit number or *nibble* and to 256 values for an eight bit *byte*. See Fig. 1-7B. For a 16-bit number, there are 2¹⁶ possible values, or 65,536—64 K for short (1024 bits = 1 K). This 16-bit length is known as an *address word* in the typical 8-bit microcomputer.

You can work out the decimal equivalent of any binary number using the method shown in Fig. 1-7C. Take the bit value (0 or 1) of each bit position and multiply by the corresponding place value; then add the result.

Table 1-3 lists decimal numbers with their binary equivalents. The range of values for a four bit number are given. The table also indicates that you can represent a binary nibble with an alternative number system, which has a base of 16. This is the hexadecimal system with which you may be familiar.

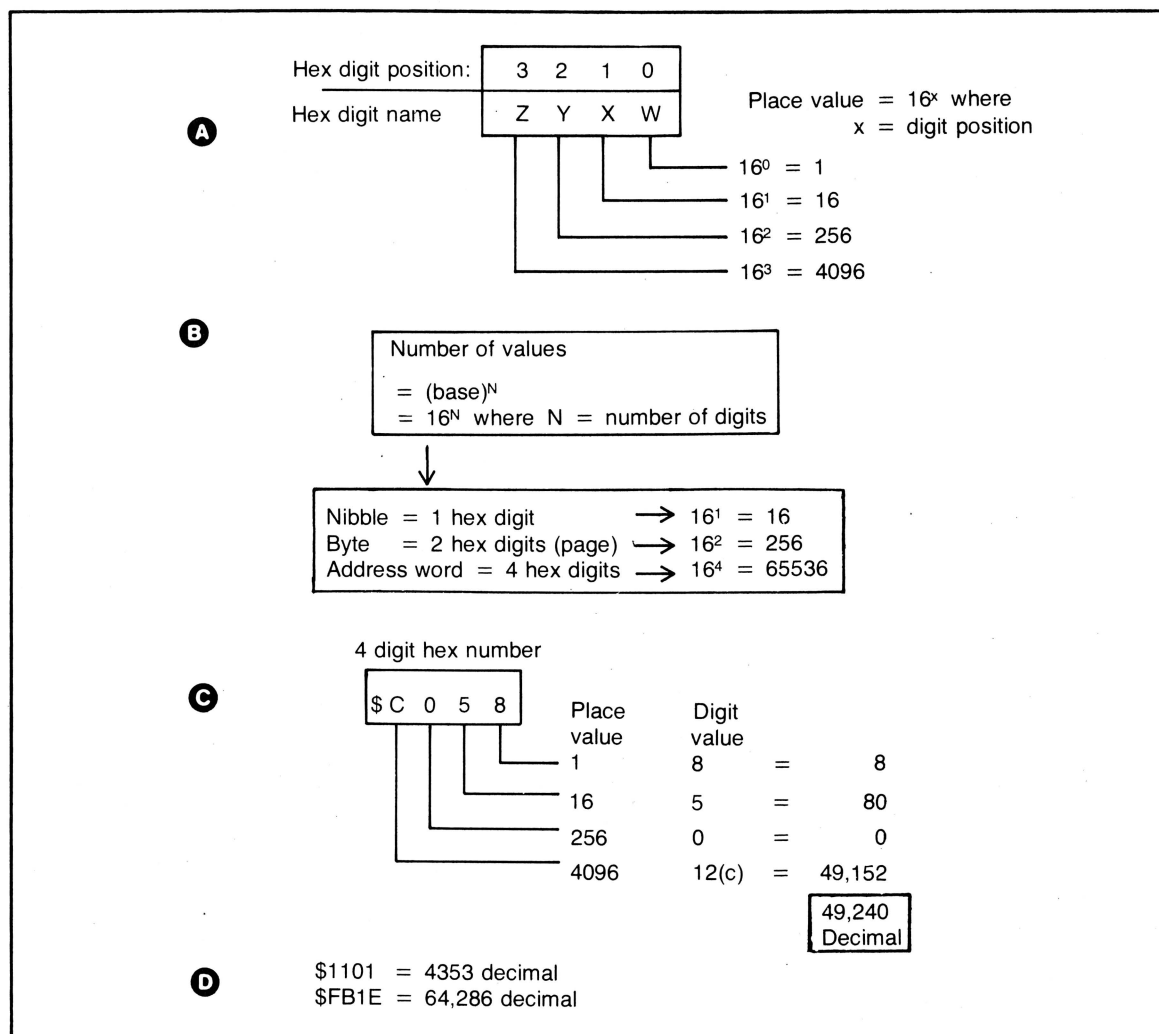


Fig. 1-8. Hexadecimal representation. Refer to the text.

Hexadecimal Representation

The reason for using the *hexadecimal* number system is simple: there is a nice symmetry between the binary system and the hexadecimal form of representation. Each 4-bit nibble in binary can be represented by a single hex digit. This is demonstrated in Table 1-3. As a convenience, you should think of the hex system as merely a condensed form of binary.

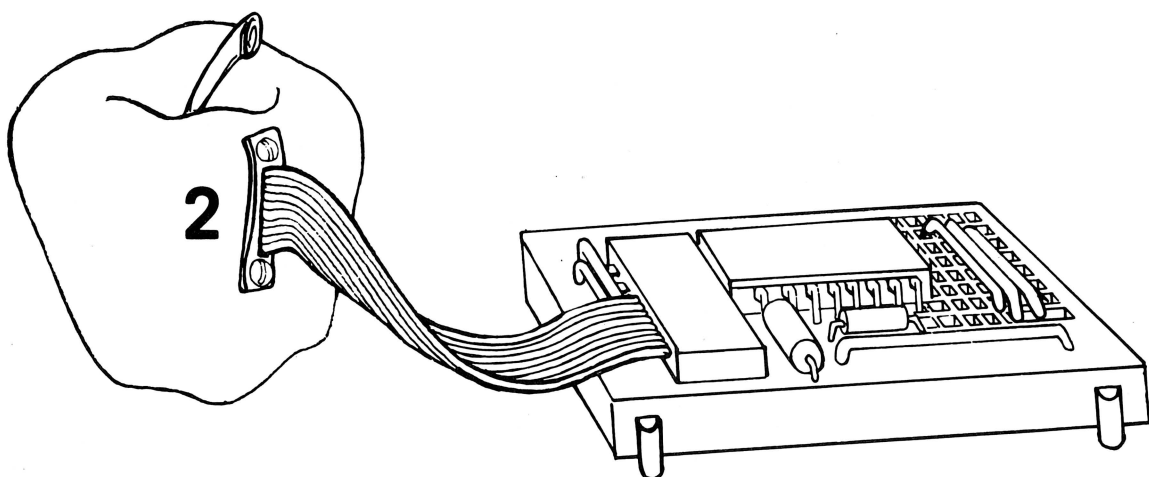
Following the same sequence as we did for binary, refer to Fig. 1-8. Again, each digit in a hex number has a position number and a position name. In this four digit number (Fig. 1-8A), we call the digits Z, Y, X, and W. The digit positions are 3, 2, 1 and 0. The place values are given by 16^x , where x is the digit position. Digit Z in the 3-position has a place value of 4096, etc.

Figure 1-8B shows that a four digit hex number can assume 65536 different values, the same as a sixteen bit binary number, certainly an improve-

ment in economy of representation. A single hex digit—with values 0 to F hex, or 0 to 15 decimal—is equivalent to a four bit binary number. Two hex digits are equivalent to a byte, and four hex digits to a 16-bit address word.

Figure 1-8C illustrates the conversion of a four digit hex number to its decimal equivalent by the same process as that for binary to decimal conversion. Note that hex numbers are preceded by a dollar sign \$. This number, \$C058, happens to be the memory address of annunciator 0 off the Apple game port (explanations later).

Finally, Fig. 1-8D shows the decimal equivalent of two other hex numbers. Verify them for yourself. The first looks the same as the binary example in Fig. 1-7C except for the preceding \$ sign. The second number happens to be the address in the Apple Monitor for reading the paddles off the game port.



The Digital Desktop Laboratory

In the last chapter, we took a broad view of digital electronics and the key role the individual IC devices play in the learning process. By now, you are probably anxious to start doing something; therefore, in this chapter you'll be shown how to set up the computer-based logic trainer. You'll also collect a few other items which, together with the trainer will comprise a desktop laboratory for studying digital devices and circuits.

First, let's look at the Apple game port.

THE GAME PORT CONNECTION

The logic trainer could be built around one of the peripheral slots on the Apple main board. With all those signals (48 to be exact) a really elaborate trainer/prototyping setup would be possible. However, slot-based input/output (I/O) projects such as these are not for the beginner. They require knowledge of interfacing, construction methods and troubleshooting, and a fair cash investment for materials and equipment.

However, the Apple game port provides the

cheapest, easiest and most immediate approach to setting up the computer-based logic trainer. It is well suited to its role because of the number of signal lines available on its socket. There are 12 signal lines plus power and ground. These are quite adequate for our logic trainer/breadboard system as well as for a number of other applications such as A/D conversion.

In short, a system based on the game port is the ideal choice for the beginner. It will give you all the necessary features of the traditional stand-alone trainer, plus the advantages of a dynamic tv monitor display of your experiments, but without the expense and experience needed for the elaborate slot-based system.

As you advance in digital electronics, you may find a need for a conventional trainer/breadboard because they are handy when several projects are going on simultaneously. Later, more sophisticated projects which require the peripheral slot signals may attract your attention. But for an outlay of 20 dollars and half hour or so to type in the utility program, you're ready to start experimenting.

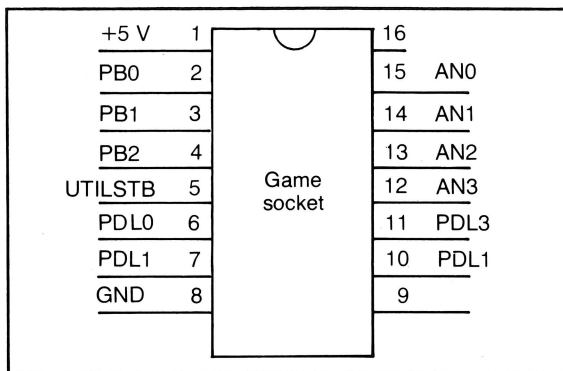


Fig. 2-1. Game socket pin-out.

Game Socket Overview

The hardware/mechanical part of setting up the trainer involves nothing more than plugging one end of a DIP (dual-in-line-package) jumper cable

into the game socket and the other end into a small breadboard strip. Before doing this, you should be very familiar with the port signal lines you will be using.

Figure 2-1 is a view of the 16-pin game socket located at the upper right hand corner of the Apple main board. Table 2-1 is a listing of all of the digital game port signals: their pin numbers, names, labels, locations in memory, and function. Also listed is the cassette input.

For the trainer you will require four outputs, four inputs, and power and ground. The four annunciator lines are 1-bit digital (on or off) outputs from the computer; these will serve as inputs to the device or circuit under study. You also need four single bit lines into the computer in order to read the outputs from the device or circuit. These are the three pushbutton lines plus the single cassette

Table 2-1. Game Port Signals, Including the Cassette Input, Used In the Computer-Based Logic Trainer. BDIS Also Uses the Utility Strobe. Paddle (Analog Game) Inputs Are Also Present. These Lines are Detailed Later on.

Pin	Signal Name	Label	Address		Function
			Hex	Decimal	
15	annunciator 0 Off	AN(0)	\$C058	49240	Output
	annunciator 0 On	AN(0)	\$C059	49241	Output
14	annunciator 1 Off	AN(1)	\$C05A	49242	Output
	annunciator 1 On	AN(1)	\$C05B	49243	Output
13	annunciator 2 Off	AN(2)	\$C05C	49244	Output
	annunciator 2 On	AN(2)	\$C05D	49245	Output
12	annunciator 3 Off	AN(3)	\$C05E	49246	Output
	annunciator 3 On	AN(3)	\$C05F	49247	Output
2	pushbutton 0	PB(0)	\$C061	49249	Input
3	pushbutton 1	PB(1)	\$C062	49250	Input
4	pushbutton 2	PB(2)	\$C063	49251	Input
rear jack	cassette in	PB(3)	\$C060	49248	Input
1	+5 volts	V _{CC}	N.A.		Nominal limit of 100mA current drain on this power supply.
8	ground	GND	N.A.		Circuit ground.

input line. Finally, you need the +5 volt power line and ground. Discussion of the paddle and utility strobe lines will come later, as they are required. Right now, let us consider the required lines in detail.

Annunciator Outputs

Figure 2-2 emphasizes the relationship between computer, annunciators and peripheral device. These four annunciator pins on the game socket serve as 1-bit outputs from the computer to the peripheral device. They can assume either of two states: on or off. The on state corresponds to a voltage high or a binary value of 1. The off state corresponds to a voltage low or a binary value of 0. When the annunciators are used to control digital circuits, it is often appropriate to think of these states as binary 1 or 0. Alternatively, they may be viewed as simply on or off, as when they are used to control a lamp, relay or alarm buzzer.

As to the actual voltages on these lines, remember that the annunciators are TTL outputs. This means that during the off/0 state the respective line has a value of near zero volts, and that during the on/1 state the respective line has a nominal voltage of around 4.0 volts. These voltage levels are typical of all TTL devices, both Schottky and nonSchottky.

Figure 2-3 indicates that TTL ICs have been designed to give well-defined output voltages. Voltage high is defined as 2.4 volts or more. Voltage low must be between 0 and 0.4 volts. Anything between 0.4 and 2.4 volts is defined as an *indeterminate output* voltage level and would imply either a faulty device or improper use of the device.

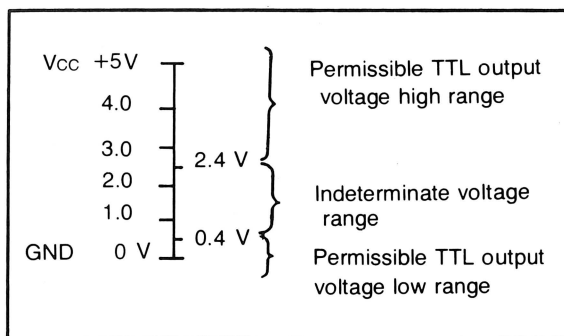


Fig. 2-3. TTL output voltage ranges off the annunciator lines.

All TTL subfamilies have been designed to these output specifications, and the annunciator circuitry, being TTL, likewise conforms to them.

Each annunciator is assigned a pair of locations in memory, one to turn it on and the other to turn it off. These locations or addresses in memory are just like switches and may be activated merely by referring to them with the proper command. The BASIC POKE command will do the job, either from within a program (deferred) or as immediate commands issued from the keyboard. Provided that you access the correct address, it makes no difference what value you poke into that address. You can even use a PEEK command. For example:

```
POKE 49240,0      or
POKE 4920,137    or
PEEK (49240)      will each cause
                  ANN(0) on pin 15
                  to go off or low.
```

```
And,
POKE 49241,0      or
```

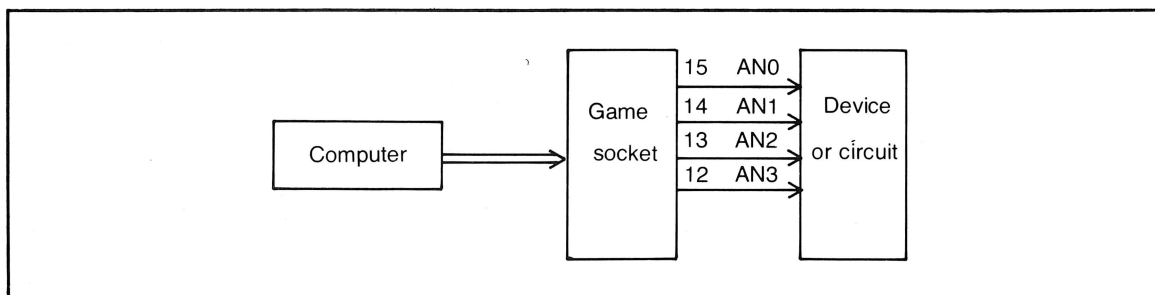


Fig. 2-2. Annunciator output lines off the game socket.

POKE 49241,217
PEEK (49241)

or
will each cause
ANN(0) on pin 15
to go on or high.

Because you are accessing one of a pair of locations, the action is much like one of the old-style two button wall switches shown in Fig. 2-4. Pressing the on button by any means gives you the same end result: the room light turns on. Now press the off button below (which popped out when you pressed on) and the light goes off. The analogy between the two annunciator addresses and the two light switch buttons is very close.

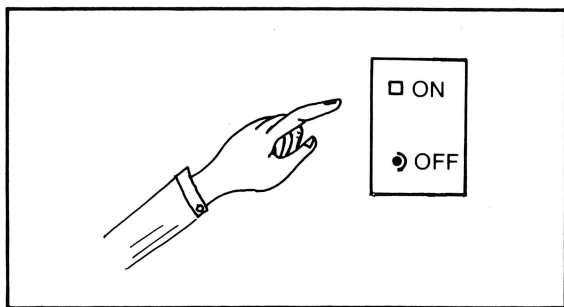


Fig. 2-4. The annunciators are toggled much like wall switches: two separate locations for turning a light on and off.

This is why the annunciator addresses and similar memory locations are referred to as *softswitches*; by accessing these locations from software you can effect a change in hardware. In this case the voltage level changes on a particular socket pin. Because you are referencing one of a pair of addresses assigned to each pin or output line, you are toggling the softswitches.

NOTE: The term *annunciator* can refer to an address in memory, a chip on the main board that holds the high or low voltage, or a particular pin on the game socket. The particular meaning is usually clear from the context. For example, *softswitch* obviously refers to the memory location, while *line* would refer to the actual pin or the wire leading from it to the peripheral circuit. The same distinctions hold true for the pushbutton inputs discussed below.

Pushbutton Inputs

Figure 2-5 illustrates the three inputs which are usually associated with the pushbuttons found on game controls. The *Apple Reference Manual* refers to these inputs as SW(0)-SW(2) and calls them switches. However, the term pushbutton input conveys the same meaning and is more specific than the term switch. Therefore PB is used as the label in this book.

Note that unlike the annunciators, these pushbutton locations are not softswitches because they are computer inputs, not outputs. The computer reads these input lines. Each pushbutton line or socket pin is assigned a single address in memory. This location will contain data indicating whether the line is in a high state or a low state. From BASIC you must use the PEEK command to find out if the voltage is high or low. The form used is:

```
x = PEEK ({address})  
x = PEEK (49249) for PB(0) for example.  
IF x > 127, then the line is high, on, binary 1.  
IF x < 128 the line is low, off, binary 0.  
x = the value in that address
```

So much for the addresses and form of command. What about voltages?

The peripheral device connected to the given PB line—whether a mechanical switch or a complex digital circuit—must place the correct voltage on that line. **Since the PB input circuitry is TTL, these voltages must adhere to TTL guidelines for acceptable input voltage ranges.**

Figure 2-6 depicts these permissible voltages. This diagram is similar to that of Fig. 2-3 except that it refers to TTL inputs, not outputs. Any voltage between 0 and 0.8 volts will be interpreted by a TTL input as a low or 0. Voltages above 2.0 volts are seen as high or binary 1.

Don't get confused about input and output. Just remember that annunciator outputs from the computer may be considered as inputs to a peripheral device under study on the breadboard. Likewise, pushbutton inputs to the computer from a

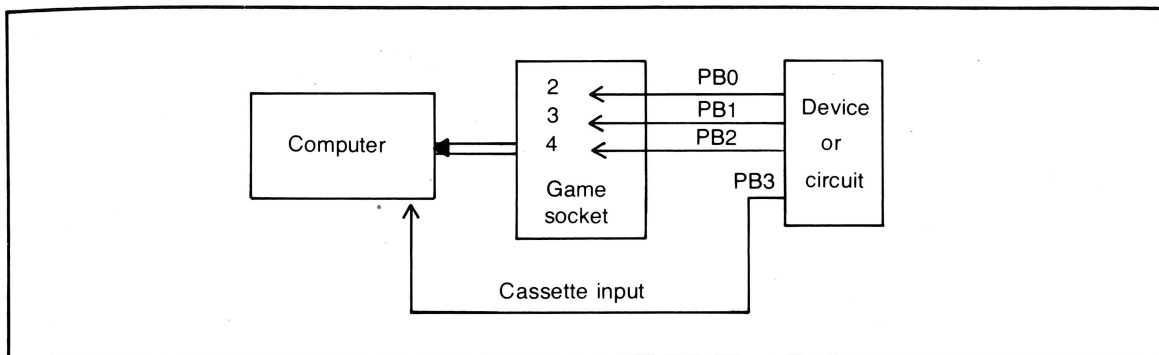


Fig. 2-5. Pushbutton input lines to the computer.

peripheral may be thought of as outputs from that device or circuit.

Caution: If you place a voltage much above +5 V. or much below 0 V. (negative) on the pushbutton inputs to the Apple, the game port circuitry reading these lines could be damaged. Since you are using the Apple's own +5 volt supply off the game port, this caution may seem unnecessary. However, at some time you may advance to a larger external power supply with multiple + and - voltages, so keep this caution in mind.

Naturally, if the peripheral is another TTL circuit, you don't have to worry about voltage compatibility. This is because the maximum allowable TTL output level for a logic low is well below the maximum allowable input level (0.4 and 0.8 volts respectively). Similarly, the minimum allowable output for logic high is well above the minimum allowable input level (2.4 and 2.0 volts respectively). This is the beauty of standardized families and subfamilies, namely compatibility of electrical characteristics. It is generally good practice to stay within the same family unless there are good reasons to the contrary, and this is another reason why LSTTL devices will be used for our experiments.

A Fourth Input

It was mentioned earlier that the cassette input could be used as a fourth digital input for

monitoring the circuit outputs under study. The physical location for this line is the cassette input jack on the back of the Apple, and is labeled as such. On the Apple *I*Ie, it is the rightward cassette symbol with the downward-going arrow. On Apple II and II+ it is labeled as IN or as CASS IN. A method which is quick, cheap and compatible with all Apple versions is to simply plug a miniature phono plug (with a single wire attached to the signal lug) into the cassette jack. The other end of the wire is stripped and connected to the peripheral circuit. That's all. There are no traces to cut or main board modifications to be made, so you don't have to be concerned about voiding your warranty.

Like the pushbutton inputs, the cassette input has its own location, as indicated in Table 2-1. To read the status on the cassette line, we use the BASIC command:

$X = \text{PEEK}(49184)$

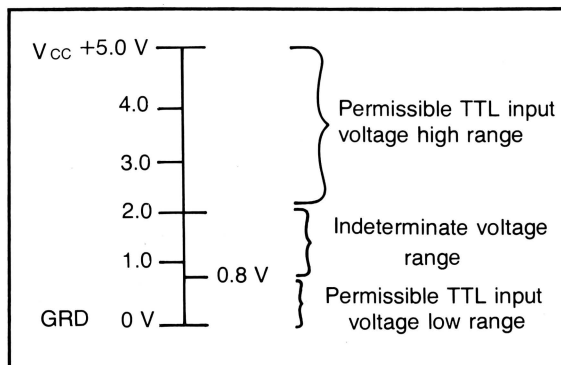


Fig. 2-6. Standard TTL input voltage levels.

Because of the circuitry involved, the voltage on the cassette input is inverted, so the corresponding values of high and low are reversed relative to the value of x . That is,

If $x > 127$ then the line is low, off or binary 0.
If $x < 128$ then the line is considered as high, on or binary 1.

Regarding the voltage levels themselves, there is no problem. The TTL output voltages from the devices being examined are compatible with the linear cassette input circuitry (essentially an IC amplifier called an op-amp).

I am going to call the cassette input pushbutton input 3, or PB(3), as a matter of convenience.

Power and Ground

Five volts at 100 milliamps (mA) current is available on pin 1 of the game socket. This means that you have 1/2 watt or 500 milliwatts (mW) of power to drive your devices or circuits. This may seem small, but it is adequate to power all of the SSI and MSI devices we will be studying, as well as a number of practical circuits consisting of several devices. Ground is on pin 8 of the game socket. Each line is connected to the respective pins on the device(s) being studied. Just remember that you should not short the +5 volt line to ground, or vice versa. If you do, the Apple's self-protecting power supply will emit a clicking sound as it attempts to reset itself by shutting on and off. Turning the machine off and checking for the inadvertent connection between +5 and GND is the best thing to do if this happens. No damage to the machine will result, but do try to avoid this mistake.

There are a few other general precautions to be taken in using the computer-based breadboard. These are mentioned in the introductory section preceding Experiment 1 at the end of this chapter. Disregarding them will do your computer no permanent harm, but do try to adhere to them.

TURNING THE APPLE INTO A DIGITAL LOGIC TRAINER

Turning the Apple into a logic trainer involves a few inexpensive hardware items by which to ex-

ternalize the gameport and cassette input signals, plus a utility program, the Breadboard-In-Software (BDIS). These are described below.

Hardware Hookup

Making the physical connection to the game socket and cassette input jack is simple. Any Apple system which has Applesoft in RAM or ROM is sufficient. Even an INTEGER machine with 16 K of memory an Applesoft in ROM will do. (Are there any left out there?) Besides the Apple, you will need these items:

- ☐ A three foot 16-conductor ribbon cable with 16-pin DIP plugs on either end.
- ☐ A solderless breadboard strip.
- ☐ A miniature phono plug and some solid hookup wire (20 gauge preferred).

Figure 2-7A illustrates the jumper cable. It consists of a length of Insulated Displacement Connector (IDC) ribbon cable, with an IDC DIP plug at either end. The term IDC refers to the mechanical arrangement of contacts in the plugs which makes the close packing of parallel conductors possible. IDC is an industry standard arrangement for jumper assemblies used in digital systems, both at the breadboard and finished product stage. You will find literally hundreds of cable and jumper arrangements advertised in the catalogs for virtually every application. Typically, the pins of the end connectors in

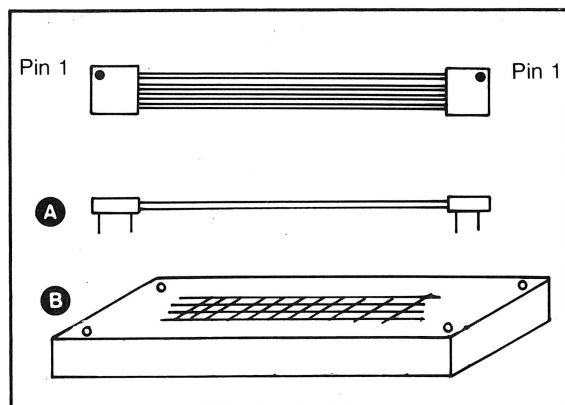


Fig. 2-7. Jumper cable and solderless breadboard.

the IDC standard, be they DIP plugs or some other type, are spaced one-tenth inch apart. (Pin 1 on the plug is indicated by the little dot in Fig. 2-7A).

This 0.1 inch standard is followed in many of the solderless breadboards commonly available, one of which is shown in Fig. 2-7B. The socket holes on these prototype boards are also on 0.1 inch centers and readily accept IC DIP packages and plugs as well as single wires. The DIP devices and connectors simply straddle the center channel in the breadboard. Since each row of five socket holes on either side of the channel are connected in common, you have four remaining holes on the side of each pin available for connections.

To keep matters simple, we will be using a widely available dual-ended 16-conductor jumper, manufactured by A P Products, and a particular breadboard strip, the Experimenter 300 manufactured by Global Specialties. The breadboard was chosen because of cost and availability, but more specifically for its expandability: there are interlocking lips on all four sides of the unit which make the addition of extra units literally a snap.

These items can be obtained from most any electronics parts distributor, either locally or by mail order. I've listed the cheapest sources I could find for these items:

☐ The 3 foot, 16-conductor ribbon jumper cable can be obtained from

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002
PHONE: 415-592-8097
Part # DJ16-3-16 \$3.59
in 1983 catalog, page 25.

Jade, Digi-Key, Active Electronics and A P Products are other sources.

☐ The solderless socket strip, or experimenter's protoboard is available from any Radio Shack outlet as well as through mail order.

Radio Shack carries the Experimenter 300 protoboard under Part # RS 276-174 for \$11.95.

☐ The miniature phono plug is also available from Radio Shack under Part # RS 274-286 @2 for \$1.29.

If you want to purchase one of the game port expanders (see Creative Computing, Sept. 82 for a comprehensive review), you will have the added flexibility that several externalized game sockets afford. They typically run from about \$20 up to \$50 for the more elaborate versions. However, you will still need the DIP jumper, breadboard and phono

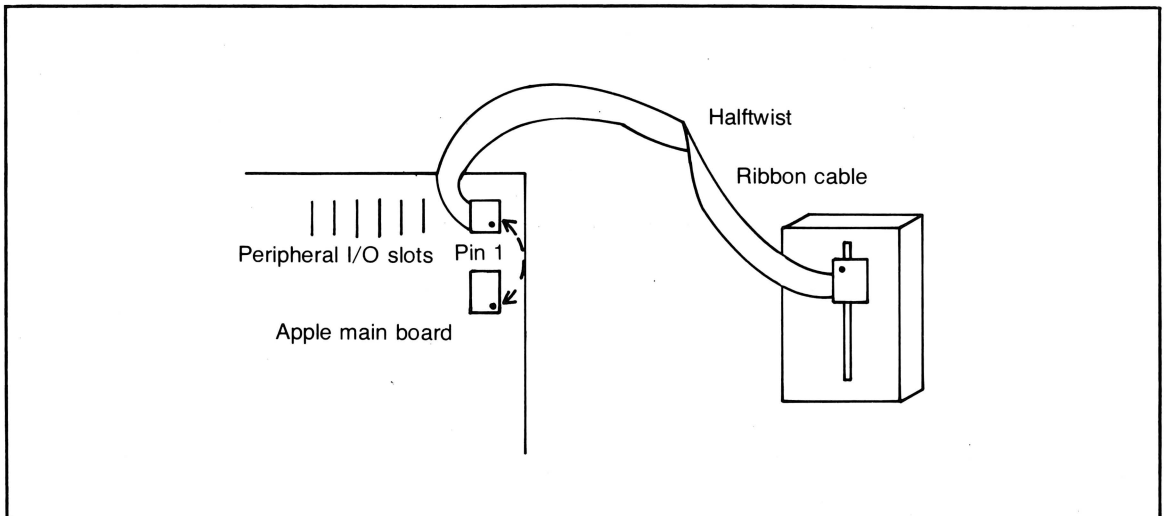


Fig. 2-8. Connecting the game socket on the main board to the solderless breadboard strip using the jumper cable. Make sure pin 1 lines up as shown at either end.

plug listed above for the experiments in this book. So unless you have the urge to play games at a moment's notice, these expanders are not necessary for our purposes.

Now, follow these steps closely. Figure 2-8 shows how the ribbon jumper is connected. With the cover off, you can locate the game socket at the upper right-hand corner of the Apple main board. Note that pin one of the game socket is at the lower right corner of the socket as you look down from the front of the machine. When you insert the DIP plug at one end of the jumper, **Take care that pin #1 of this plug is aligned with pin 1 of the game socket!** A half twist in the cable will be necessary to align pin 1 as indicated in the figure.

Now lead the jumper out of the back panel slot of the Apple (or through a panel cutout in Apple IIe) and replace the cover. Plug the other end of the ribbon cable into one end of the breadboard strip as shown in Fig. 2-8. This time the 1 pin of the DIP plug will be oriented to the upper left, with the plug straddling the center channel of the breadboard, as illustrated.

Finally, take a 2½ to 3 foot length of solid 20 gauge hookup wire and strip about ½ inch of insulation off each end. Unscrew the phono plug cap. Pass one bare end of the wire through the tip of the lug hole (or wrap the bare end if there is no hole) and solder it as shown in Fig. 2-9. Crimp the lug teeth around the insulation and screw the cap back on. Then plug the phono plug into the cassette input jack at the back of the Apple. You can put the other bare end of the lead wire into an empty hole on the solderless protoboard if you desire.

The Breadboard-In-Software Program

Every logic trainer has to fulfill the following basic functions:

- ☐ Send digital signals to a device or circuit under examination.
- ☐ Receive signals from the circuit.
- ☐ Display those signals.

The conventional, stand-alone logic trainer has slide or toggle switches to send highs or lows to the circuit. These inputs to the circuit or device are indicated by the physical position of the switches. Light emitting diodes (LEDs) usually serve as indicators for the circuit outputs. The only way of recording the various states of inputs to and outputs from the circuit is by paper and pencil. One must be sure that the right switch position and the right LED condition correspond to the right column of data you are writing on. This method of keeping track of experiments can be rather laborious to say the least. It can take a bit of the fun out of working with the hardware.

In the computer-based trainer, things are much easier. The keyboard digits replace the toggle switches, and the tv or video monitor replaces the simple row of LEDs. With a computer generated display of our experiments, the work of examining device and circuit operation is almost a pleasure when compared to the conventional approach. Both inputs and outputs are echoed to the screen in a convenient, easy to read format, one which you can modify as appropriate. You are able to use meaningful headings, such as IC pin-outs and signal

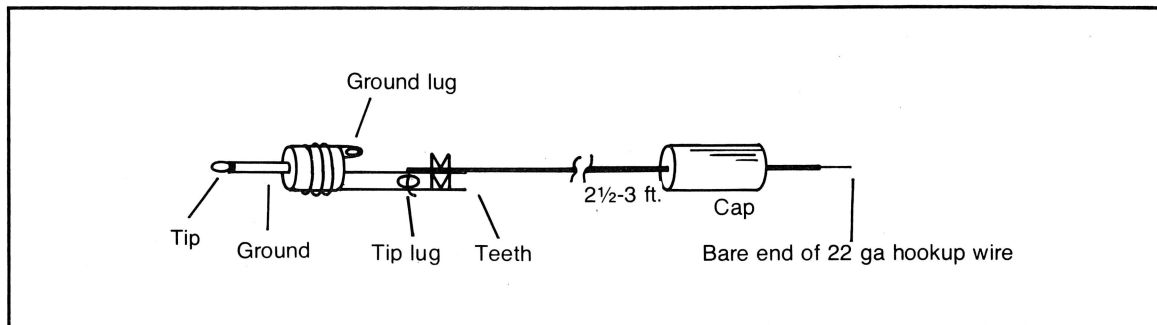


Fig. 2-9. Details of the cassette input jack with attached lead wire.

Table 2-2. The Three Hardware Items Needed to Turn the Apple Into a Digital Logic Trainer.

DIP Jumper	A three foot, 16-conductor ribbon cable with 16-pin DIP plugs at either end. Used to externalize the game port signals. A P Products 924116-36
Solderless Breadboard	Usually, a rectangular plastic object with many small socket holes or "tie points" for the connection of wires and components. Cheap and reusable. A boon to the experimenter and circuit designer. Global Specialties: Experimenter-600
Miniature Phono Plug	Used to input signals into the "fourth pushbutton" line, the cassette input. Any local or mail order store.

names, both as an aid in double checking the wiring, and in interpreting the final results. In fact, you will be able to generate a table of results and print them out for a permanent record of your experiments.

All of these features are provided by the Breadboard-In-Software utility program, and are further outlined in Table 2-2. Each is explored in detail in Experiment 1 at the end of this chapter. As you begin using BDIS, you will discover that it has something of the feel of a simple video game, or more accurately of a simple spreadsheet program.

BDIS, the brain of the trainer, is written mostly in Applesoft BASIC as you can see from

Listing 2-1. A short Keyread machine-language subroutine is also necessary to monitor keyboard input and to aid in continual updating of the display. This is given in Listing 2-2. I'll talk about entering the listings in a moment.

BDIS was written in a structured manner. Most of the code is devoted to parsing using input, formatting the display, reconfiguring the display, and enable printout. Only three lines of code are actually devoted to toggling the annunciator outputs and reading the pushbutton inputs. The program will run on any standard Apple: the II with A/S (Applesoft) in ROM, on a II+, and on a IIfx in the

Table 2-3. The Main Features of the Breadboard-In-Software (BDIS, C. 1984).

Four Line Heading	Two are user definable for meaningful signal names, pin-out designations, etc.
Tabular Display	Up to four inputs and four outputs displayed to the screen. Format can be reconfigured by the user.
Automatic Line Numbering	Each entry line is numbered. This feature is especially convenient when studying such devices as counters and encoders.
Commands	<ul style="list-style-type: none"> - for toggling the annunciators on and off. - to generate a new entry line and "truth tables". - to clear the screen (HOME). - to reconfigure the screen display (see Experiment 1). - to "dump" the screen display to a printer. - to quit BDIS. - to trigger the utility strobe, pin 5
Number Keys	Apple II with Applesoft in ROM, II+ or IIfx.
Return	Apple IIfx owners should have the 80 column card turned off before running the program.
Ctrl-H	
Ctrl-R	
Ctrl-P	
Ctrl-Q	
"S"	
Compatibility	

```

10 REM *****
15 REM BREADBOARD IN SOFTWARE
20 REM BDIS - VERSION 1.1
25 REM COPYRIGHT 1984
30 REM CHARLES ENGELSHER
35 REM *****
36 REM
37 REM
40 PRINT CHR$(12); CHR$(21): REM CTRL-L&U TO HOME AND DEACTIVATE 80
   COL CARD ON JCE.
50 POKE 33,40: REM IN CASE YOU'VE BEEN EDITING WITH POKE 33,33.
60 PRINT CHR$(4);"BLOAD READKEY,A$300"
100 REM
101 REM >>>>>>> MAIN <<<<<<<
102 REM
105 HOME : VTAB 10: HTAB 9: PRINT "BREADBOARD IN SOFTWARE"
110 GOSUB 1010: GOSUB 1200: REM INITIALIZE & PRINT HEADINGS.
120 VTAB 5: GOSUB 5015: REM NEXTLINE
200 CALL 768:KY = PEEK (6):TS = SW: REM READ KEYBOARD & STORE PRIOR SW
   VALUE.
205 IF KY = 0 THEN GOSUB 3010: GOSUB 4020: GOTO 200: REM CONTINUE UPDA
   TING PUSHBUTTON DISPLAY UNTIL A KEY IS PRESSED.
210 SW = KY - 48: IF (SW = ) 0) AND (SW ( = NAN) THEN GOSUB 2010: GOSUB
3010: GOSUB 4010: GOTO 200: REM IF SW VALUE IS LEGAL.
215 SW = TS: REM IF NOT, RESTORE PRIOR (LEGAL) SW VALUE.
220 IF KY = 13 THEN GOSUB 5010: GOTO 200: REM 'CR'...NEXTLINE
230 IF KY = 17 THEN GOTO 900: REM Q^...QUIT
240 IF KY = 8 THEN HOME : VTAB 5: GOSUB 5015: GOTO 200: REM H^..'HOME'
   , IE CLEAR SCRNL.
250 IF KY = 18 THEN HOME : GOTO 6010: REM R^...RECONFIGURE
260 IF KY = 16 THEN GOSUB 7010: GOTO 200: REM P^..PRINT SCRNL
300 FOR I = 1 TO 10:X = PEEK ( - 16336): NEXT : GOTO 200: REM ...KEYPR
   ESS ERROR SOUND
800 REM .....ONERR HANDLER.....
810 X = PEEK (218) + PEEK (219) * 256:Y = PEEK (222): REM LINE# & ER
   R CODE
820 PRINT "ERRCODE=";Y;"---LN#=";X
830 GOTO 900
900 POKE 34,0: POKE 35,24: END
1000 REM
1001 REM >>>>>>> INITIALIZE <<<<<<<
1002 REM
1010 IN$ = "0":KY = 48:SW = 0:LB = 2:LN = 0: REM INPUT/LABL/LN# VARS
1015 POKE - 16368,0: REM CLEAR KYBD STR $C010 FOR READKEY.
1020 ABA = 49240:PBA = 49249: REM BASE ADDRS OF ANN & PB.
1030 DIM LAN(3,1),LPB(3),VAN(3),VPB(3): REM LOCATIONS & VALUES.
1040 DIM TITLE$(3),IHEAD$(3,3),OHEAD$(3,3),TEMP$(7): REM HEADING ARRAYS.
1050 NAN = 3:NPB = 3:BL$ = " ":DSH$ = "-": REM # OF COLUMNS & DUMMY $'S
   .
1060 DIM SCREEN(23,38): REM IS USED ALONG WITH V,H & B IN PRINT SCREEN-
   7000.
1100 FOR I = 0 TO 3: FOR J = 0 TO 1:LAN(I,J) = ABA:ABA = ABA + 1: NEXT
   J,I

```

```

1110 FOR I = 0 TO 2:LPB(I) = PBA + I: NEXT I:LPB(3) = 49248: REM CASS.I
    N = 4TH PB
1120 FOR I = 0 TO 38:BL$ = BL$ + " ":DSH$ = DSH$ + "-": NEXT I
1130 FOR I = 0 TO 3: READ TITLE$(I)
1140 FOR J = 0 TO 3: READ IHEAD$(I,J):L = LEN (IHEAD$(I,J)):IF L = 2 THEN
    IHEAD$(I,J) = " " + IHEAD$(I,J)
1145 IF L = 1 THEN IHEAD$(I,J) = " " + IHEAD$(I,J) + " "
1150 NEXT J: REM ABOVE IS FOR CENTERING/JUSTIFICATION.
1155 FOR J = 0 TO 3: READ OHEAD$(I,J):L = LEN (OHEAD$(I,J)):IF L = 2 THEN
    OHEAD$(I,J) = " " + OHEAD$(I,J)
1160 IF L = 1 THEN OHEAD$(I,J) = " " + OHEAD$(I,J) + " "
1165 NEXT J,I
1170 VTAB 24: HTAB 1: INVERSE : PRINT "COMMANDS";: NORMAL
1175 PRINT "..'S'..RTN..H^..R^..P^..Q^..";
1180 POKE 35,23: HOME : RETURN
1200 REM
1201 REM >>>>> PRINT HEADINGS <<<<<<
1210 FOR I = 0 TO 3: VTAB (I + 1): HTAB 1: CALL - 868: PRINT TITLE$(I)
    ;":": REM -868=CLEOL.
1230 FOR J = NAN TO 0 STEP - 1: PRINT " ";IHEAD$(I,J);: NEXT J: PRINT
    " ";
1240 FOR J = NPB TO 0 STEP - 1: PRINT " ";OHEAD$(I,J);: NEXT J
1250 NEXT I: PRINT : CALL - 868
1260 PRINT LEFT$ (DSH$, (NAN + NPB + 2) * 4 + 7)
1270 VTAB 6: HTAB 1: POKE 34,5: RETURN
2000 REM
2001 REM >>>> TOGGLE ANNUNCIATOR <<<<
2002 REM
2010 VAN(SW) = NOT VAN(SW): REM TOGGLES VALUE
2020 POKE LAN(SW,VAN(SW)),0: REM SETS LS259
2030 RETURN
3000 REM
3001 REM >>>> READ PUSHBUTTONS <<<<
3002 REM
3010 FOR I = 0 TO 2:VPB(I) = PEEK (LPB(I)) > 127: NEXT I:VPB(3) = PEEK
    (LPB(3)) < 128: REM CASSETTE INPUT INVERTS SIGNAL.
3020 RETURN
4000 REM
4001 REM >>>>>>> DISPLAY <<<<<<<
4002 REM
4010 HTAB (9 + 4 * (NAN - SW)): PRINT VAN(SW);: REM DISPLAY ONE ANN.
4020 FOR I = NPB TO 0 STEP - 1: REM DISPLAY ALL PB'S.
4030 HTAB (14 + 4 * (NAN + NPB - I)): PRINT VPB(I);: NEXT I
4040 HTAB (8 + 4 * (NAN - TS)): PRINT CHR$ (32);: REM ERASE CURSOR FRO
    M LAST POSIT.
4045 HTAB (8 + 4 * (NAN - SW)): FLASH : PRINT CHR$ (32);: NORMAL : REM
    PLACE IT AT NEW POSITION (MAY BE UNCHANGED).
4050 RETURN
5000 REM
5001 REM >>>>>>> NEXTLINE <<<<<<<
5002 REM
5010 HTAB (8 + 4 * (NAN - SW)): PRINT CHR$ (32);: REM ERASE CURSOR, REA
    D CURRENT LINE POSITION.
5015 X = PEEK (37): REM CURRENT LN#.
5020 IF X < 21 THEN LN = X - 4
5030 IF X > 20 THEN LN = LN + 1
5040 IF LN > 999 THEN LN = 0
5050 HTAB 40: PRINT : HTAB (3 - (LN > 9) - (LN > 99)): PRINT LN;

```

```

5060 FOR I = 0 TO NAN:VAN(I) = 0: POKE LAN(I,0),0: HTAB (9 + 4 * I): PRINT
"0";: NEXT I: PRINT " :";: REM SET ANN'S=0 & PRINT.
5070 SW = 0: GOSUB 3010: GOSUB 4020: REM POSITION CURSOR, READ&DISPLAY PB
'S
5080 RETURN
6000 REM
6001 REM >>>>> RECONFIGURE <<<<<
6002 REM
6010 HOME : HTAB 12: PRINT "RECONFIGURATION": HTAB 17: PRINT "MENU": PRINT
6020 PRINT "A...CHANGE # OF COLUMNS"
6030 PRINT "B...REASSIGN LABELS"
6035 PRINT "C...RETURN TO MAIN (BREADBOARD)"
6040 PRINT : PRINT : FLASH : PRINT ">>>>>";: NORMAL
6045 GET IN$
6050 IF IN$ = "A" THEN GOTO 6100
6060 IF IN$ = "B" THEN GOTO 6200
6065 IF IN$ = "C" THEN HOME : GOSUB 1210: VTAB 5: GOSUB 5015:X = FRE
(0): GOTO 200
6070 FOR I = 1 TO 10:X = PEEK ( - 16336): NEXT : GOTO 6010
6099 REM .....CHANGE # OF COLUMNS.....
6100 HOME : PRINT "HOW MANY DEVICE INPUTS (#ANN) AND": PRINT "OUTPUT (#
PB) COLUMNS DESIRED?": PRINT : PRINT
6110 INPUT "#ANN(1TO4), #PB(1TO4)";NAN,NPB
6120 IF NAN < 1 OR NAN > 4 OR NPB < 1 OR NPB > 4 THEN CALL - 1051: CALL
- 1051: GOTO 6100
6130 NAN = NAN - 1:NPB = NPB - 1
6140 GOSUB 1210: GOTO 6010: REM PRINT HEADINGS & RETURN TO RECONFIG. ME
NU.
6199 REM .....REASSIGN LABELS.....
6200 NAN = 3:NPB = 3: GOSUB 1210: REM FULL WIDTH HEADINGS.
6210 HOME : PRINT : PRINT "REASSIGN LABL1 OR LABL2 (1 OR 2)": PRINT "(
PRESS SPACE BAR TO RETURN TO MENU)": FLASH : PRINT ">>>>>";: NORMAL
6230 GET IN$:KY = ASC (IN$) - 48:LB = KY + 1: HOME
6235 IF IN$ = " " THEN GOTO 6010
6240 IF KY < 1 OR KY > 2 THEN CALL - 1051: CALL - 1051: HOME : GOTO
6210
6250 INVERSE : PRINT "LABL";KY;: NORMAL : HTAB 7: INPUT ">";IN$
6260 IN$ = "": FOR I = 1671 TO 1702:IN$ = IN$ + CHR$ ( PEEK (I)): NEXT
I: REM READ LINE 6.
6280 FOR I = 0 TO 7:TEMP$(I) = MID$ (IN$,4 * I + 1 + (I > 3),3): NEXT
6290 HOME : HTAB 7: FOR I = 0 TO 3: PRINT " ";TEMP$(I);: NEXT I: PRINT
":";
6300 FOR I = 4 TO 7: PRINT " ";TEMP$(I);: NEXT I: PRINT : FLASH : PRINT
DSH$: NORMAL : PRINT
6310 PRINT "PRESS (Y)ES TO REASSIGN ABOVE TEXT TO": PRINT "LABL";KY
6320 PRINT "PRESS ANY OTHER KEY TO ABORT THE": PRINT "REASSIGNMENT."
6330 GET IN$: IF IN$ = "Y" THEN GOTO 6400
6340 HOME : VTAB 6: HTAB 8: PRINT "REASSIGNMENT CANCELLED": FOR I = 1 TO
1000: NEXT I: GOTO 6010
6400 FOR I = 0 TO 3:IHEAD$(LB,I) = TEMP$(3 - I):OHEAD$(LB,I) = TEMP$(7 -
I): NEXT I
6410 GOSUB 1210: GOTO 6010: REM PRINT HEADINGS/RETURN TO MENU
7000 REM
7001 REM >>>>> PRINT SCREEN <<<<<
7002 REM

```

```

7010 V = PEEK (37) + 1:H = PEEK (36) + 1: REM SAVE CURSOR POSIT.
7020 FOR I = 1 TO V: VTAB I: CALL - 990:B = PEEK (40) + 256 * PEEK (
41): FOR J = 0 TO 38:SCREEN(I,J) = PEEK (B + J): NEXT J,I
7030 PRINT CHR$ (13); CHR$ (4);"PR#1"
7035 PRINT CHR$ (9);"20L"
7037 PRINT CHR$ (9);"80N"
7040 FOR I = 1 TO V: FOR J = 0 TO 38: PRINT CHR$ (SCREEN(I,J));: NEXT
J: PRINT : NEXT I
7050 PRINT CHR$ (4);"PR# 0"
7060 VTAB V: HTAB H: RETURN
9000 DATA GPSIG,AN0,AN1,AN2,AN3,PB0,PB1,PB2,PB3
9010 DATA GPIN#,15,14,13,12,2,3,4,CS
9020 DATA LAB1,A,B,C,D,W,X,Y,Z
9030 DATA LAB2,0,1,2,3,0,1,2,3

```

Listing 2-1. The bulk of the BREADBOARD IN SOFTWARE (BDIS) program is in Applesoft. It is best to type it in exactly as it appears.

```

*PR#0      1 *****
:ASM      2 *
          3 *          READKEY          *
          4 */          FOR BDIS          *
          5 *          *
          6 *****
          7
          8 KEY      =      $C000
          9 CLRKEY   =      $C010
         10 SPKR     =      $C030
         11 UTLSTB   =      $C040
         12 WAIT     =      $FCAB
         13
         14          ORG  $300
         15
0300: AD 00 C0 16          LDA  KEY
0303: 8D 10 C0 17          STA  CLRKEY          ;RESET BIT 7 OF KEY.
0306: C9 D3      18          CMP  #$D3          ;NEG ASCII FOR 'S'.
0308: F0 0A      19          BEQ  UTILOUT
030A: C9 80      20          CMP  #$80          ;KEYPRESS?
030C: 90 18      21          BCC  NOTPRSD        ;NO.
         22
030E: E9 80      23          SBC  #$80          ;YES. CONV TO POS. ASCII.
0310: 85 06      24          STA  $06          ;VAR. "KY" FOR BDIS.
0312: B0 16      25          BCS  EXIT          ;CARRY REMAINS SET IF A
         26 *          KEY WAS PRESSED.
         27
0314: AD 40 C0 28          UTILOUT LDA  UTLSTB          ;MUST READ NOT WRITE.
0317: A2 04      29          LDX  #4
0319: A0 19      30          LDY  #25
031B: AD 30 C0 31          SOUND  LDA  SPKR          ;SOUND FOR UTILITY STROBE.
031E: 98        32          TYA
031F: 20 A8 FC 33          JSR  WAIT
0322: 88        34          DEY
0323: CA        35          DEX
0324: D0 F5      36          BNE  SOUND
0326: A9 00      37          NOTPRSD LDA  #0          ;CODE FOR NO KEYPRS.
0328: 85 06      38          STA  $06          ;KY VARIABLE.
032A: 60        39          EXIT      RTS

```



```
--End assembly--
```

```
43 bytes
```

```
Errors: 0
```

Listing 2-2. The small READKEY machine language program should be BSAVED to disk with the command BSAVE READKEY, A\$300, L\$100. Use an assembler with this listing.

forty column mode. Line 40 was included for the Apple IIe versions.

If you examine the BDIS program BASIC listing, you'll note that it has several sections: main, initialize, print screen headings, toggle annunciator, read pushbuttons, update screen display, nextline, reconfigure, and print screen. By reading through the program, you should get some sense of the overall logic behind it. Comments have been included in the form of REM statements where appropriate.

An obvious question arises. Since the BDIS program is predominantly BASIC, is it fast enough? The answer is yes. As presented here, BDIS has all the essential functions that you would demand of a logic trainer, and it responds promptly. However, if you need simultaneous updating and display of a few dozen inputs while complex keyboard commands are parsed and executed, then you would have to graduate to a machine language program. Requirements of this sort however are in the more sophisticated realm of commercial logic testers, not digital trainers.

Entering BDIS

When entering the Applesoft portion BDIS from Listing 2-1 be sure to follow the code line by line. Do not change line numbers. Enter each line and its associated command(s) exactly as the listing indicates. You may also be tempted to omit the REM statements, believing that speed of execution will be noticeably improved. Don't, because it won't.

The reason for these cautions is simple. If you've made an error in entry, it is a lot easier to find your mistakes from an exact (or nearly exact) listing by comparing it to Listing 2-1. Also, you may want to examine the program later from a printout of BDIS, if you have a printer. Having the complete

program in front of you, on which to make notes and otherwise mark up, is more convenient.

PRINTER NOTE: The screen dump to printer routine (7000 Block in BDIS), is written with the assumption that your printer card is in slot 1. Change Line 7030 as appropriate if your card is in a different slot. Line 7035 is an instruction to the commonly used Grappler Card (Orange Micro Inc.) to set the left margin to column 20. Use the command appropriate for your printer card. If you do not have a printer card, a screen dump command (Ctrl-P) will cause the computer to lock up. Reset the computer to continue.

When you have finished typing in BDIS, save it to an initialized diskette. You can make BDIS the program which first runs when the disk is booted from a cold start. Such a program is called the *Hello* program. Follow the instructions in the DOS manual on the use of the Master Create program. This program is on the system master disk, and will make BDIS (or any other specified program) the one to run on boot up. Thus, you'll have BDIS up when you turn on the Apple, assuming you have an Apple II plus or IIe.

Entering Readkey

You can enter the machine language input routine, Readkey, by one of three methods:

- ☐ By means of a commercial assembler.
- ☐ By the Mini-Assembler which is part of the Apple II and IIe monitor ROM.
- ☐ By direct entry using the monitor, something possible with all Apples, regardless of version or type of machine.

If you have a full-blown commercial assembler

program (e.g., Lisa, Big Mac, to name two among many), you can utilize the assembler listing given in Listing 2-2.

```

0300- AD 00 C0 LDA $C000
0303- 8D 10 C0 STA $C010
0306- C9 D3 CMP #$D3
0308- F0 0A BEQ $0314
030A- C9 80 CMP #$80
030C- 90 18 BCC $0326
030E- E9 80 SBC #$80
0310- 85 06 STA $06
0312- B0 16 BCS $032A
0314- AD 40 C0 LDA $C040
0317- A2 04 LDX #$04
0319- A0 19 LDY #$19
031B- AD 30 C0 LDA $C030
031E- 98 TYA
031F- 20 A8 FC JSR $FCAB
0322- 88 DEY
0323- CA DEX
0324- D0 F5 BNE $031B
0326- A9 00 LDA #$00
0328- 85 06 STA $06
032A- 60 RTS

```

Listing 2-3. The READKEY program for use with the Apple Mini-Assembler.

If your Apple has the Mini-Assembler (standard on IIs and IIses), then use the listing in Listing 2-3. The use of the Mini-Assembler is detailed in the Apple Reference Manual that came with your

```

0300- AD 00 C0 8D 10 C0 C9 D3
0308- F0 0A C9 80 90 18 E9 80
0310- 85 06 B0 16 AD 40 C0 A2
0318- 04 A0 19 AD 30 C0 98 20
0320- A8 FC 88 CA D0 F5 A9 00
0328- 85 06 60

```

Listing 2-4. The READKEY program for entering directly via the monitor.

machine. See pages 49-51 in the Apple II Reference Manual, or pages 110-114 in the *IIs* Manual.

You can, if you desire, directly enter the code for Readkey by invoking the Monitor (Call-151 from BASIC), and typing in the two digit hex numbers as given in Listing 2-4. Pages 48-49, and pages 107-108 in the II and *IIs* Manuals provide details on direct Monitor entry of machine code.

THE REST OF THE LOGIC LAB

Figure 2-10 and Table 2-4 provide a concise summary of the rest of the logic lab. As a convenience, the individual digital ICs to be used in the first set of IC experiments in Chapter 3 are included in Table 2-4. Mail order suppliers of chips, tools, and equipment are given in Table 2-5.

Briefly stated, you will need the following items to complete your logic lab:

- ☐ Several IC chips with which to start working.
- ☐ An IC tool to safely insert and remove them from the breadboard.
- ☐ An inexpensive voltmeter to observe their output pin voltages, as well as for other routine measurements.
- ☐ A TTL data manual.

IC Devices and Tools

The five low power Schottky (LSTTL) chips listed in Table 2-4 are not, of course, all you will require for the experiments in this book. However, they will see you through the next two chapters; no other components are necessary until Chapter 5. Other components needed for later experiments are listed in the Materials section at the beginning of each experiment.

Note that in Fig. 2-10A, pin 1 on a typical DIP IC package is indicated by a dot or notch at one end. If a notch alone is used as a marker, pin 1 is at the top left corner with the device oriented as shown. As you would expect, the pins are on 0.1 inch centers, which is the same as the breadboard hole spacing.

The end view in Fig. 2-10A illustrates how the pins flare out on unused chips. It is necessary to

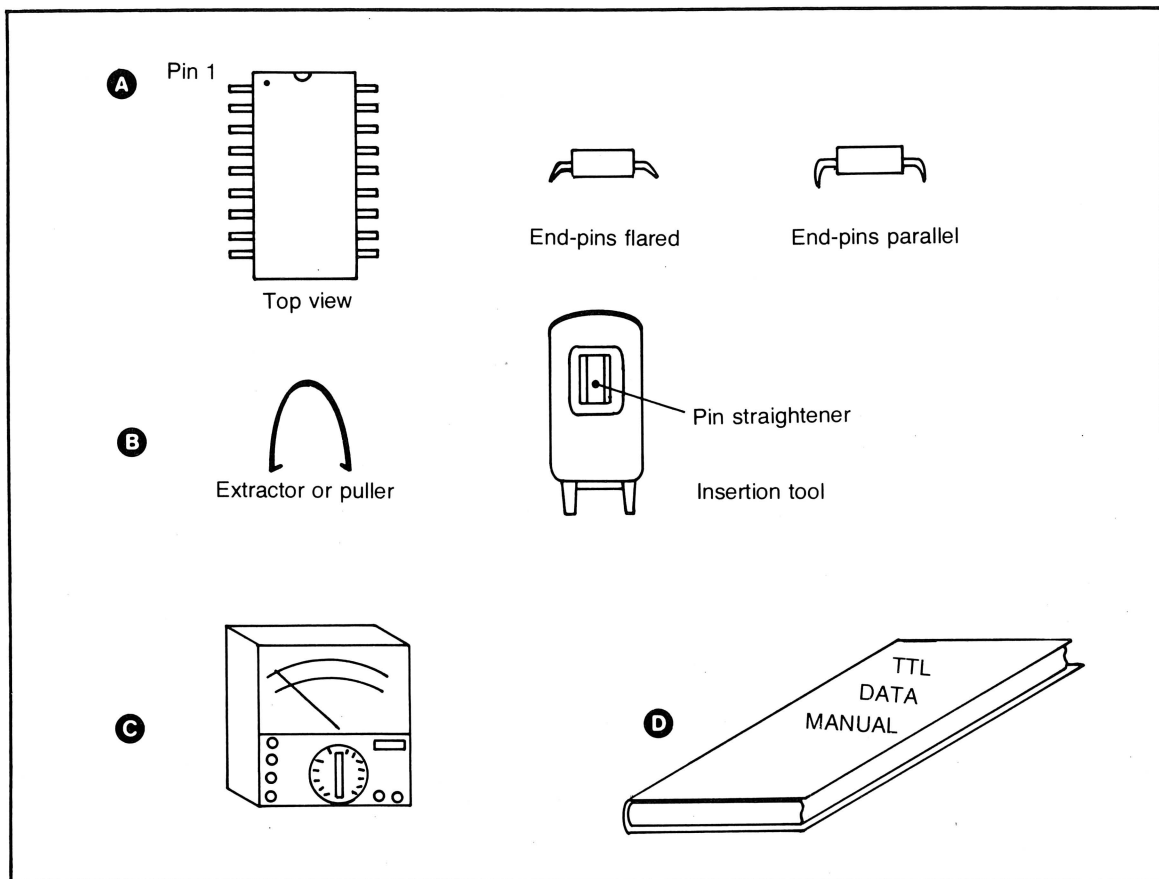


Fig. 2-10. Other components needed to complete the logic laboratory: LSTTL chips, IC tools, VOM or DMM, and TTL data manual.

make them parallel or nearly so, as shown, before attempting to insert them into a socket or breadboard. A means of safely extracting the device is also needed. Bending the pins by turning the chip on each side in turn and exerting downward pressure is one way to make them parallel. The chip can later be removed by prying each end gently out of its position. Occasionally, you will ruin a chip by snapping off a pin if you apply these methods with any regularity.

A better way is to use the tools specifically designed for insertion and extraction, as shown in Fig. 2-10B. The extractor or *IC puller* is basically a pair of thin metal tongs with small teeth at the end for grabbing on to the IC. The insertion tool allows you to pick up the chip and safely push it into the

socket, with pins parallel. Both items should be obtained. Radio Shack markets a package which contains both extractor and insertion tool at a very reasonable price. By the way, the pin straightener on the inserter is for pins bent too far inwards.

Voltmeter

A voltmeter is an essential tool on any electronics workbench. The least expensive of these are known as VOMs. These are little hand-held units with a panel meter and a selector switch for measuring ac and dc voltage, resistance and sometimes current, depending on the model. They can be had for as little as \$15 to \$20 and are sufficient for our needs.

More expensive digital voltmeters (DVMs) or

Table 2-4. The Other Major Elements of the Digital Desktop Laboratory.

Items	Description	Sources
Low Power Schottky * Integrated Circuits	1-74LS00 2 Input Quad NAND 1-74LS02 2 Input Quad NOR 1-74LS04 Hex Inverter NOT 1-74LS08 2 Input Quad AND 1-74LS32 2 Input Quad OR	Any of these items may be obtained from mail order firms or from local stores such as Radio Shack. See Table 2-5
IC Tools	IC Extraction Tool IC Insertion Tool	OK Machine and Tool: Part #s EX-1 and INS-1416 or both from Radio Shack: Cat. # 276-1574
Voltmeter **	KRISTA 30B-150 DVM \$54.95 B&K Precision 2845 DVM \$79.95 (full featured) RADIO SHACK DVM 22-197 \$59.95 or VOM 22-201 \$19.95 The TTL DATA BOOK for Design Engineers	EPS Box 5356 Berkeley CA 97405 800-227-0104 FORDHAM 260 Motor Parkway Hauppauge, NY 11788 800-645-9518 Any local outlet
TTL Data Manual	TTL DATA BOOK	TEXAS INSTRUMENTS Box 225012 Dallas, TX 75265 214-995-6531 FAIRCHILD INSTRUMENT CORP. 464 Ellis Street Mountain View, CA 94042 415-962-5011 SIGNETICS CORP. Box 409 Sunnyvale, CA 94086 408-739-7700
	TTL LOGIC DATA MANUAL	

Discrete Components These are listed in the individual experiments. For the early experiments, only a few 1/2 or 1/4 watt resistors are needed, in the 330 to 1000 ohm range. An assortment pack (RADIO SHACK) of 100 resistors will fulfill most needs nicely.

* **Important !!** Do not use standard TTL, as these use several-fold more power than low power Schottky devices. Make sure you are buying LS TTL.

** Prices on the DVMs and VOM are current as of this writing. There are many retail and mail order sources available, so shop around.

Table 2-5. Additional Suppliers of Parts, Tools and Equipment.

Albia Electronics
P.O. Box 1833
New Haven, Conn. 06508
Phone: 1-800-243-6953

Carries a line of affordable test equipment, and prototyping boards.

Active Electronics
133 Flanders Rd.
Westborough, Mass. 01581
Phone: 1-800-343-0874

Good source of digital and analog ICs. Wide variety of LSTTL chips, and discrete components. Prototyping and test equip. also.

Global Specialties
P.O. Box 1942
New Haven, Conn. 06509
Phone: 1-800-243-6077

Another good source of reliable but inexpensive test equip. and prototyping boards.

Jade Computer Products
4901 W. Rosecrans Ave.
Hawthorne, Calif. 90250
Phone: 1-800-421-5500

Large mail order electronics firm.

OK Machine and Tool Corporation
3455 Conner St.
Bronx, New York 10475
Phone: 212-994-6600

Jumpers, proto boards, and wire wrap tools and supplies.

digital multimeters (DMMs) are an alternative. (DVM and DMM are synonymous.) The problem with VOMs is that they *load* or draw significant current from the circuit or device being measured. Just how significant this loading effect is depends on the resistance of the circuit and the range being measured. The net effect of this loading is a somewhat inaccurate reading. This problem is resolved by *high input impedance* instruments whose resistance is so high that they draw almost zero current from the circuit being tested. Their input resistance or impedance is on the order of several million ohms. They draw as little as a microamp or less in low voltage measurements. DVMs are therefore highly sensitive and quite accurate. Modern DVMs feature a digital readout, quick response, multiple functions, and sometimes autoranging.

A 20,000 ohms-per-volt VOM will be adequate here. If you can afford a bit more, I suggest a digital voltmeter. As you can see from Table 2-4, some models are in the \$50 to \$70 dollar range. The extra expenditure is worth it as a DVM will give you more

accurate results with greater convenience and provide years of trouble free service.

TTL Data Manual

The last item on the list is a TTL data manual. These books contain all the technical information that you would ever need (or want) to know about the TTL ICs from a given manufacturer. At least one such manual is essential. You will consult it often to verify numeric designations, pin-outs, logic functions, power consumption and other characteristics. Manuals from Texas Instruments, Signetics and Fairchild Semiconductor are recommended, as listed in Table 2-4. You should seriously consider Texas Instruments' *TTL Data Manual for Design Engineers* in its hardcover version as it will stand up well under the heavy use it will receive. The companies mentioned have a wealth of technical material, including applications handbooks containing practical advice in using IC devices and designing useful circuits. Most are fairly liberal in their distribution policy, especially if you

are a student or a potential user of their products.

Sources of this technical literature, besides the manufacturers themselves, are as follows:

- ☐ Mail order companies.
- ☐ Over the counter parts stores, retail.
- ☐ College bookstores, especially those associated with schools offering technical/engineering courses.
- ☐ Large bookstores, Barnes and Noble and B. Dalton in particular.

INTRODUCTION TO THE EXPERIMENTS

All of the experiments in this book follow a regular and fairly conventional format, consisting of a stated purpose, a list of materials, a detailed procedure section, and a discussion/summary. There is generally more hand-holding in the earlier experiments, including pictorial schematics of the breadboard, and detailed descriptions for hookup of components and the use of BDIS. This explicitness is appropriate, even given the relative simplicity of the materials involved.

There are a few things that you should not do when hooking up circuits on the computer-based breadboard. Let's take them in order.

Don't short VCC + 5V to ground. This is obvious. If you do inadvertently and momentarily short VCC +5 volts to ground (pins 1 and 8 on the game connector), the Apple computer's switching power supply will reset the machine. If this short is not momentary, there will be a steady clicking sound as the power supply recycles its resetting sequence. Normally, no harm is done in either case, as the power supply was designed for just such mishaps. Merely turn the machine off, check out where the short is, and resume as usual.

Don't connect any annunciator output to VCC to ground. Ever. This might result in excessive current drain on the annunciator output chips. In the Apple II and IIe, these chips can be replaced if damaged. The II/II+ chip is cheap and available in any well stocked parts store, while the IIes custom I/O chip is much more expensive and must be special ordered. In either case, take care to avoid

connecting the annunciator output pins directly to the +5 volt or ground lines.

These first two precautions apply to every version of the Apple. The next two precautions apply specifically to the Apple IIe.

On the Apple IIe, do not press the open Apple or closed Apple functions keys while BDIS is in operation. This may create a brief short between +5 V and ground, resulting in an automatic reset. Again, no harm done, simply an inconvenience.

Also, on the IIe, defer any hookups to the PB0 and PB1 lines until after the Apple is turned on. Occasionally, a circuit may source too much current through these lines when the Apple is first turned on, again resulting in recycling of the power supply. Wait until the Apple has been turned on before making the connections to pins 6 and 10. If you do want to ground a PB input for any reason (such as in the checkout procedure in Experiment 1), do so through a 220 ohm to 1,000 ohm resistor. This current limiting *pull-down* resistor prevents excessive current flow. The same applies to connecting any PB line to +5 volts (logic high). Use a current limiting *pull up* resistor in the 220 ohm to 1000 ohm range.

Don't in any way be dismayed by these cautions. If you take care in following the experimental instructions, you will not have such minor mishaps. The precise basis for a few of these cautions may be a little obscure to you, but follow them anyway. Their rationale will be clear by the end of the electronics section (Chapters 5-7).

EXPERIMENT 1, SETUP

Purpose

In this section, you will make sure that BDIS performs as it should, and examine its various features with particular attention to configuring the screen display. You will observe TTL output voltages using the voltmeter.

Materials

The computer-based trainer (Apple II with BDIS utility, jumper and solderless breadboard).

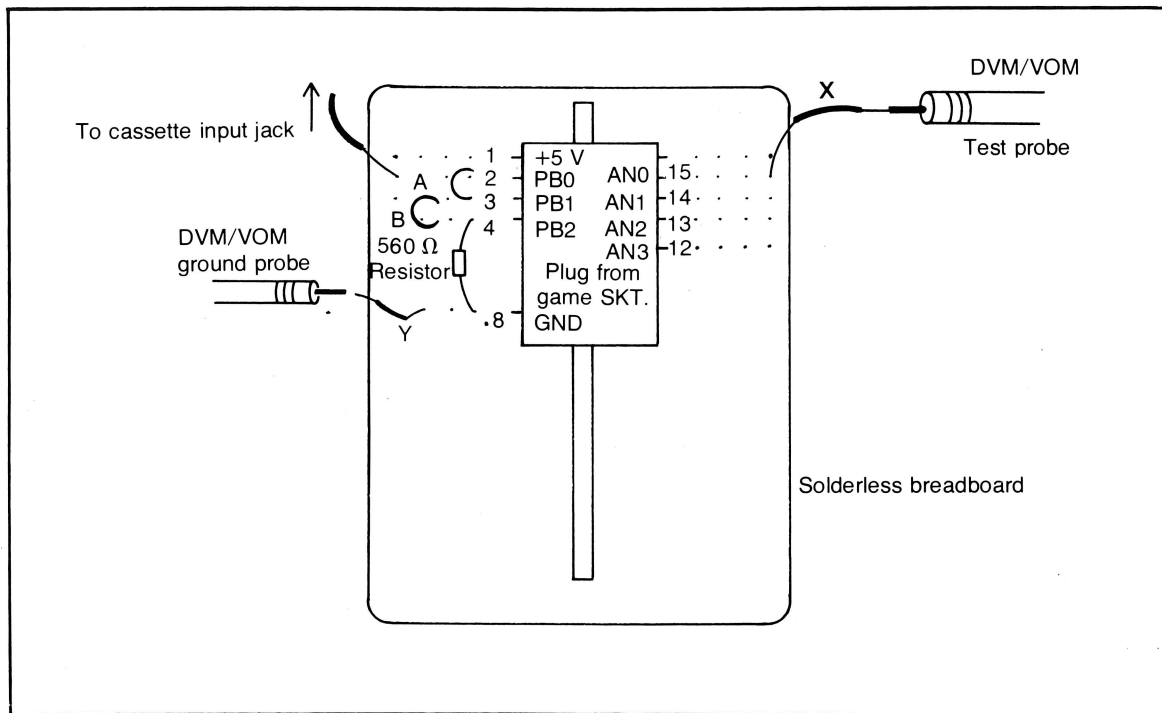


Fig. 2-11. Experiment 1, set up. Refer to the text.

VOM or DMM/DVM

20 gauge hookup wire and wirecutters/ strippers.

Note: these items will be assumed in all other experiments.

One 560 ohm $\frac{1}{2}$ or $\frac{1}{4}$ watt resistor (acceptable range from about 220 ohms to about 1000 ohms).

Procedure

1. With the computer off and the jumper plugged into one end of the breadboard, as shown in Fig. 2-11, cut two short jumper wires each about one inch long. Strip about $\frac{1}{4}$ inch off each end. Insert one jumper across socket pins 2 and 3, and another across pins 3 and 4. This is done by simply inserting the bare ends of the wire into one of the four holes that line up with the corresponding pin, as indicated by wires A and B in the figure. You can connect the bare end of the cassette input lead (the fourth pushbutton) to pin 2 or 3 or 4. (Do not attach the 560 ohm resistor to ground yet).

2. A lead 4 to 6 inches long, labeled X, should be prepared by stripping $\frac{1}{2}$ inch or so off the ends. Insert one end into pin 15 and leave the other end free. This free end, to which you might want to attach a small alligator clip, provides a point of contact for the *test lead* of the voltmeter. Likewise, prepare a similar lead, Y, and attach it to pin 8 to provide contact with the *ground lead* of the voltmeter.

3. Turn on the computer. Attach one end of the current limiting pull-down resistor to pin 2 or 3 or 4, and the other end to ground pin 8. Now all PB inputs are effectively grounded through this resistor. Run BDIS (which will boot automatically if you've made it the Hello program) and observe the screen. The four line heading will look like that in Table 2-6, with only line zero on the display. The top two header lines are for game port signal names and their corresponding game port pin numbers. The second two lines are user definable LABL1 and LABL2, to which default values have been assigned when BDIS is first run. The line at the very bottom

Table 2-6. Truth Table for Experiment 1.

GPSIG:	AN3	AN2	AN1	AN0:	PB3	PB2	PB1	PB0
GPIN#:	12	13	14	15:	CS	4	3	2
LABL1:	D	C	B	A :	Z	Y	X	W
LABL2:	3	2	1	0 :	3	2	1	0
<hr/>								
0	0	0	0	0 :	0	0	0	0
1	1	1	1	1 :	0	0	0	0
2	0	0	0	0 :	1	1	1	1

of the display is a list of the commands, which are used for generating entry lines, obtaining a print-out, and so forth. RTN allows you to generate another data line on the screen. H[^](Ctrl-H) clears the screen, like a HOME command. Each will be illustrated below. R[^](Ctrl-R) initiates the reconfiguration mode, by which the screen display and headings can be reformatted by the user. P[^](Ctrl-P) is the screen dump to printer command. Q[^](Ctrl-Q) is to quit or exit to BASIC. The S (the letter S, not the control character) is for the *utility strobe off* pin 5, something to be discussed in one of the later experiments.

4. Observe that the four annunciator outputs are all zero, as this is the way the program was initialized. The four pushbutton columns should all read zero, because they are all connected to ground.

5. If the display headings or the values of the annunciators and pushbuttons do not correspond to the appearance of Table 2-6, (four heading lines plus one entry line, line 0) check the hookup and then check the listing of BDIS. Check such things as proper pin orientation and connections first, as this is the quickest. Then double-check your entry of BDIS line by line.

6. Measure the annunciator TTL output voltages using the X lead. Your voltmeter should be in a low dc voltage range (to allow for a 5 volt reading) with the test lead attached to wire X and the ground lead attached to the wire labeled Y. Move the X lead

from pin 15 through pin 12 and take successive measurements. The annunciator outputs will measure in the range of 0.0 to 0.4 volts, typical of an unloaded TTL output in a low or binary 0 state.

7. Press return to generate entry line, 1 in Table 2-6. Toggle the annunciators so that they all turn on (binary one on the screen) by pressing the numbered keys on the keyboard 0 to 3. The display will now look like Table 2-6 with lines 0 and 1 only on the screen.

8. Measure the annunciator voltages off pins 12 through 15. This time the values will be in the 3.0 to 4.0 volt range, typical of a TTL high or binary 1 state. Actually, something very near 4.0 volts should be obtained for these unloaded outputs.

9. Press return to generate line 2. Note first that the annunciators have again been set to zero, because this is the way BDIS sets up the next line of data as you press return. Now, change the pull-down resistor so that one end is attached to the +5 volt power supply pin 1, and the other end is connected to any one of pins 2, 3, or 4. It is not necessary to turn the computer off when changing this wire alone. Now a voltage high is present on the four pushbutton inputs to the computer. Note that this is echoed immediately to the screen display.

10. Your screen should now look exactly like that in Table 2-6 if you have followed the above steps.

11. If you have a printer, issue a screen dump

Table 2-7. The Reconfiguration Command.

A									
GPSIG:	AN1	AN0:	PB0						
GPIN#:	14	15:	2						
LABL1:	B	A :	W						
LABL2:	1	0 :	0						

0	0	0 :	1						
B									
GPSIG:	AN2	AN1	AN0:	PB1	PB0				
GPIN#:	13	14	15:	3	2				
LABL1:	C	B	A :	X	W				
LABL2:	2	1	0 :	1	0				

0	0	0	0 :	1	1				

command with control P, to get hard copy of the results. Again, as previously mentioned, BDIS assumes your printer card is in slot one. If not, change line 7030's PR#1 to correspond to your printer slot.

12. Now you can clear the screen with a control H. Alternatively, you can restart with a control Q to quit followed by a run; this will bring you back to the screen appearance you started with in step 3 above. Try both methods.

13. Use the reconfiguration command, control R, in order to change the number of columns displayed. Press Ctrl-R and follow the menu option A.

Type in 2,1 in response to the prompt "How many inputs/outputs?" Return to main with the C command. Compare with Table 2-7A. Repeat the process again: Ctrl-R / A / 3,2 / C. The results should look like those shown in Table 2-7B.

14. Use the reconfiguration command, control R, again in order the reassign the headings. Type B in response to the menu, and 2 in response to the prompt for which label. Enter a line of dashes followed by return. Then press Y in response to the prompt for assignment and return to main with C. The sequence of entry in shorthand form looks like:

Ctrl-R / B / 2 / "-----etc." / Y / C.

15. The display should look like that in Table 2-8. Note that the continuous dashed line is broken up into triplets of three dashes, each falling under the three-character column labels above them. BDIS removes anything typed on the seams between the columns. Whatever new labels you reassign to the various columns in either or both fields (LABL1 and/or LABL2) are therefore limited to three characters. This will be adequate for pin numbers and signal names. If you run off the line during entry, BDIS will again simply ignore the extra characters.

16. One other feature you may have stumbled upon is the repeat or clock function. If you hold a valid number key (0-3 if four annunciator input columns are on display) and the repeat key down at the same time, the corresponding annunciator will rapidly alternate between 0 and 1. Try it. Of course, Apple IIe owners can simply hold down the

Table 2-8. Relabeling Example (see step 8).

GPSIG:	AN3	AN2	AN1	AN0:	PB3	PB2	PB1	PB0	
GPIN#:	12	13	14	15:	CS	4	3	2	
LABL1:	D	C	B	A :	Z	Y	X	W	
LABL2:	---	---	---	---	---	---	---	---	

0	0	0	0	0 :	1	1	1	1	

selected key and it will automatically toggle the annunciator on and off.

Summary

In this experiment you have gained some familiarity with the use of BDIS by trying out the various commands and configuration options:

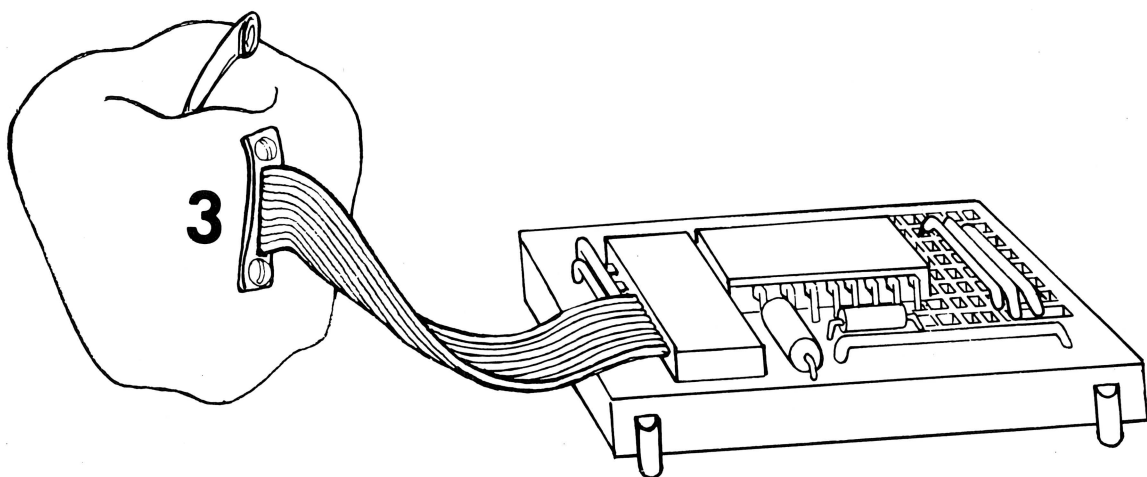
- ☐ Toggle annunciators with keyboard number keys.
- ☐ Return to generate a new entry line for creation of truth tables, about which we will say more later in the next chapter.
- ☐ Clear table with (Ctrl-H, denoted by H[^]).
- ☐ Reconfigure display by changing the number of input or output columns, or by reassigning either or both of the two user headings. This is denoted by R[^]. Print to screen with Ctrl-P, denoted by P[^]. Quit BDIS with Ctrl-Q, denoted by Q[^]. Again, we'll deal with the utility strobe later.

In following the above exercise, you have probably noticed that the shorthand method of describing the configuration of BDIS in steps 13 and 14 is fairly simple. We will start using this shorthand designation in future experiments under the general set-up portion of the procedure.

You have observed TTL output voltage levels with the voltmeter, and should be more comfortable with the game socket labels and pin numbers.

If you made any mistakes in entry, you also noticed that BDIS has fairly good error trapping at all levels of user input. Still, if you mess up a tabular display in later work which you require as hard copy, use H[^]ome, Q[^]uit and run.

Automatic line numbering is also provided. (What happens if you exceed 18 lines?) Play around with BDIS a bit more if you want; you can't hurt anything. Then proceed to the next chapter on basic logic functions.



The Six Basic Logic Functions

In this chapter, we are concerned with the functional characteristics of digital integrated circuits. Six logic functions will be examined. They have not been chosen arbitrarily. They are considered basic because all other digital devices, circuits, and systems are constructed from these building blocks. By handling and studying the chips that incorporate these functions you will, in a very real sense, be exposed to digital electronics in miniature.

You will learn the basic combinational logic functions and demonstrate them on the logic trainer that you have just assembled. You will be introduced to such things as truth tables, logic equations, schematic symbols, and to the concept of *gate equivalence*. Virtually all of the material in the remainder of the book is based at least in part on the concepts and methods mentioned in this chapter. So proceed at a comfortable pace in order to fully understand what follows.

Obtain the low power Schottky ICs and other components listed in Table 2-4 so that you will be ready for Experiment 2.

COMBINATIONAL SSI DEVICES

The small scale integration (SSI) devices which we will be examining are combinational devices. In Fig. 3-1A and 3-1B, a combinational device is contrasted with a sequential device. The difference is that the outputs of a combinational device depend solely on the input(s) at any given moment while the outputs of a sequential device depend on prior outputs as well as on the current inputs. The sequential device has a feedback loop of sorts that carries information about the current state of the device back into the input side. This information now becomes part of any new inputs as they occur and therefore influences the resultant output. In a sense, sequential logic elements are able to remember prior conditions.

An example of a simple combinational device would be one in which the single output is high or binary one if both inputs are the same, and low or binary zero if both inputs are different. A more complex combinational function would be that of a decoder that converts binary code (zeroes and

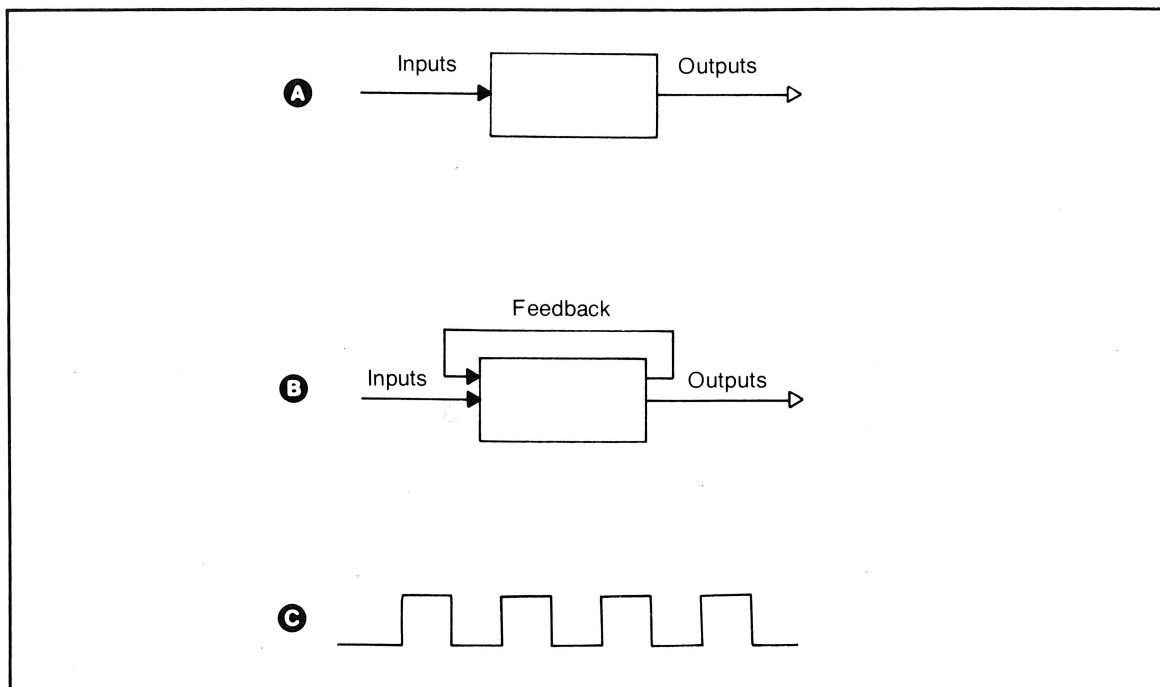


Fig. 3-1. (A) Combinational and (B) sequential functions. (C) Sequential (clock) output.

ones) into the more familiar decimal. In neither case does it make any difference what the previous state was. The output will be high or low in the first example depending upon what the current state of the two inputs. In the case of the binary to decimal converter or decoder, the decimal output depends upon the current binary input, not on prior inputs.

A simple sequential device would be a clock or oscillator; the single output alternates between high and low and might produce an output signal like the square wave shown in Fig. 3-1C. In this case, the output does depend upon what the prior output was. If it were high, the next output state must be low after a certain interval. How quickly the output alternates between the two states is determined by circuit values. An example of a more complex sequential device would be a counter. In a typical decimal counter, a single input signal pulses the device through a number of states, sometimes in a repeating cycle. It may count from zero to nine, incrementing once for each input pulse, going back to zero after reaching nine, and so on.

Of course, most practical digital circuits consist of both combinational and sequential elements. For instance, a digital clock counts time in binary, as you would expect from a digital circuit, but displays the results in readable form for the convenience of its human owners. It has a sequential clock function as well as combinational decoding circuitry for the display.

Sequential devices are more complex than combinational devices because of their memory of and dependence upon prior output states. But the two types of logic are not fundamentally all that different. In the final analysis, sequential devices consist of nothing more than a number of combinational devices piggybacked together in a circuit. This is another reason why a knowledge of SSI combinational logic is being stressed.

The Physical Package

The term *physical package* means more than just the shape of the digital IC and to the fact that it may have two parallel rows of pins (in the case of

DIPs: dual-in-line packages). This is because as an IC user, you will also be interested in the logic functions provided in the IC package and in its particular subfamily and electrical characteristics. You will also want to know what signals are assigned to which pins. This information is given, in a sort of shorthand, in the designation number on the top of the chip.

A typical DIP IC package is illustrated in Fig. 3-2A. This one is the SN74LS04N hex inverter. Let me explain the parts of this alphanumeric code, beginning with the series number.

The hex inverter in Fig. 3-2A is a TTL digital IC package, and is referred to by part number SN74LS04N. Many TTL packages begin with this 74 prefix. In fact, the 7400 series (74XX) designation is an industry standard for identifying TTL integrated circuits. It was originally introduced by Texas Instruments and was adopted by other IC manufacturers such as Signetics and Fairchild. 7400 series devices are intended for consumer grade products which, while quite sophisticated, do not

have to meet the stringent operating conditions of certain industrial, research, and military applications. Devices which must satisfy these latter demands are manufactured according to more exacting standards and carry a 5400 series designation (or 54XX).

The two-letter code in the middle of the designation stands for the low power Schottky subfamily of TTL, with all that this implies in terms of speed, power, loading characteristics and other electrical properties. Other subfamilies with different tradeoffs among these properties are listed in Fig. 3-2B.

The last two numbers in the designation refer to the function performed by the logic device. In this case, 04 refers to signal inversion. Each inverter takes a voltage at the input and reverses its value. A high input becomes a low output, and conversely, a low input becomes a high at the output. That is, a binary zero (0) becomes one (1) and a binary one is converted into a zero.

Two other letter codes should be mentioned.

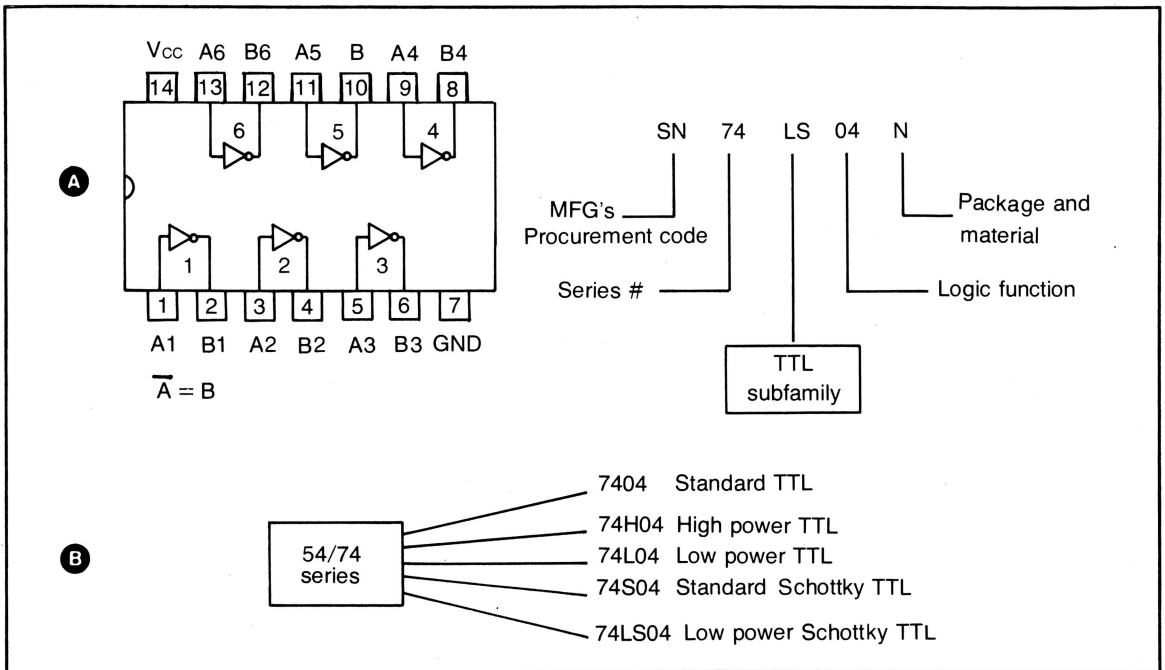


Fig. 3-2. Pin-outs and series numbers are part of the description of the physical package.

The prefix SN refers to the manufacturing and testing guidelines followed for the particular package. SN means that standard criteria have been used; the JAN prefix refers to military procurement standards; RSN means that the package was radiation hardened. Letter suffixes refer to the shape and material of the package. For example, N is a plastic DIP, J is ceramic, and T is a metal flat-pack.

Most consumer grade packages are the standard plastic DIPs and have the prefix SN and the suffix N. (These last little details bother the heck out of some people, so I mention them briefly to relieve the suspense. Pick up any TTL data manual if you have to know more about prefix, suffixes, MIL-specs and other such matters.)

Finally, note that each package has a pin-out: there are assigned pin numbers for inputs, outputs, power, and ground. In this example, each inverter has one input and one output pin. Pin 1 is the input pin for the first inverter; pin 2 is its output. In the 74LS04, there are six inverters per package. With power and ground pins this comes to a total of 14 pins, typical of many SSI chips. Note that all five subfamilies perform the same logical operation (signal inversion) and that the pin-outs of any '04 hex inverter, whatever the subfamily, are also the same.

The Functional Description

In order to understand what a digital device actually does, you have to use a variety of descriptive methods. Some forms of representation, such as logic equations, are short and elegant, but obscure to the beginner. Others, such as *truth tables*, fully detail all the input and output states of the device, but are more cumbersome to write down. Other forms of description have their own advantages and drawbacks. All of the descriptive approaches are required because they are complementary to one another. The descriptive tools that we will use to describe digital devices throughout the book include the following:

- ☐ A semi-schematic of the IC package, like the one in Fig. 3-2.
- ☐ A schematic symbol of the logic element

itself, of which there may be more than one per package.

☐ A logic or Boolean equation relating output states to input states. Boolean methods are explained in Chapter 4.

☐ A truth table, which also relates output to input.

☐ A timing or signal diagram.

☐ A brief verbal statement about what the device does.

Again, the LS04 hex inverter will be used to illustrate the discussion. Figure 3-3 shows the different descriptive methods for the LS04 IC. The six inverters, all performing the Boolean NOT function, are shown inside the package in Fig. 3-3A. This pin-out type of representation has already been covered.

Figure 3-3B isolates one of the inverters. To the left, input, output, power and ground are labeled as one would see in a conventional diagram. Often one sees the label 1/6 LS04 to signify how many devices are being used in a circuit. This facilitates chip count. To the right, the simplified schematic symbol is drawn. In a schematic diagram, it is not unusual to leave out the power and ground connections.

Figure 3-3C shows the logic equation for the NOT function. It reads, "output B is the inverse of input A." The bar over A indicates logical inversion. In Boolean equations, input quantities are to the left and outputs are to the right. As far as possible, one tries to maintain this left to right convention in drawing the schematic symbols as well. This is not always possible in complex circuits, however.

In the truth table in Fig. 3-3D, the inputs are again to the left, and the outputs are to the right. Truth tables are less concise than equations, but they spell out each and every possible state of the device. Each line across the table refers to one unique state of the device. In this case, there are only two input conditions, A=0 and A=1, so there are only two lines. If there were more inputs, the number of possible states would multiply geometrically. For example, two inputs would give a total of

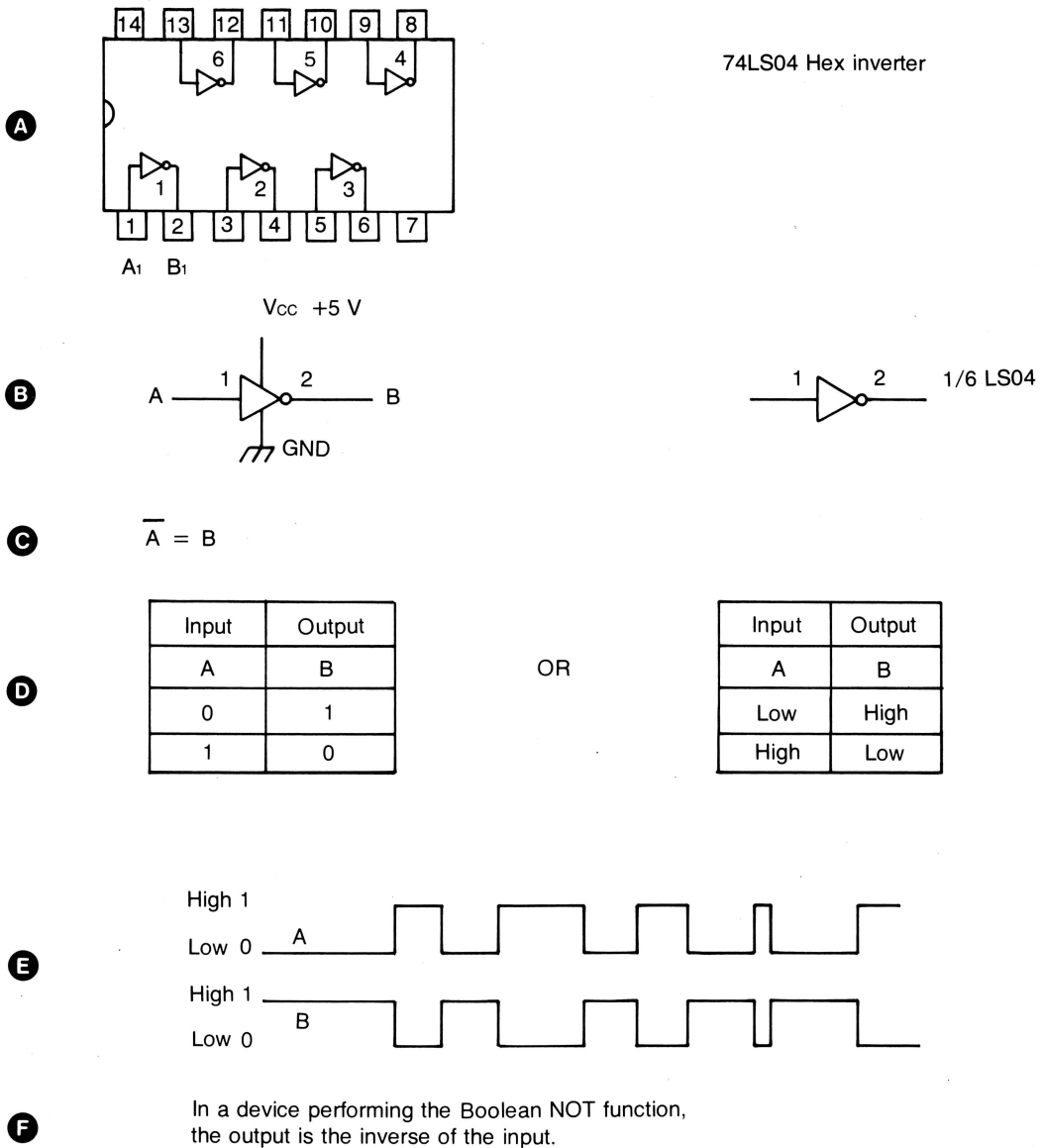


Fig. 3-3. The black box description is a functional one, supported by logic symbols, Boolean equations, truth tables, timing diagrams and verbal descriptions.

four possible states, three inputs eight, four inputs sixteen, and so on. The general rule is

$$\text{Number of Possible States} = 2^N,$$

where N is the number of inputs.

In practice, a set of equations for a given circuit is often derived from the state-by-state description provided in the truth table. The equations are then manipulated by certain rules into a more simplified form. Then, a schematic diagram can be drawn from the equations. The truth table is often the first step in designing a digital circuit. After the circuit has been constructed it provides a reference source during the debugging phase. The relationship between these elements will be explored in depth in the next chapter.

Another descriptive method is the timing or signal diagram. This graphic representation of the inverter function is shown in Fig. 3-3E. For the simpler combinational functions, such diagrams are usually unnecessary. However, timing diagrams are extremely useful in analyzing sequential logic and complex systems where precise timing of operations is essential. The process of fetching an address, an instruction and data, and then executing the command in your computer is a prime example of such timed, complex operations. System timing diagrams are key elements in the description of these sequences.

Last, there is the verbal description in Fig. 3-3F. This is just as important to an understanding of the simpler devices as the more technical looking descriptions. You cannot hope to express functions of a microprocessor in twenty-five words or less, of course. But you can very accurately describe small and even medium scale device operation in a few short sentences. Such verbal statements are useful when you are trying to analyze a digital schematic, or when you are writing down general specifications for a circuit that you may be designing.

TTL Handling Precautions

TTLICs are rugged items. They can tolerate a fair amount of thermal, electrical and mechanical (vibrational) abuse because certain safety margins

have been designed into them. Still, reasonable care should be taken when connecting them in circuits. The easiest way to understand the practical precautions is to think of each set of pins—input, output, power and ground—as having its own dos and don'ts. Let's consider these three sets of pins and their associated precautions.

Power and Ground (Fig. 3-4A). Always identify these pins first. In many 14-pin SSI packages, the +5 volts is assigned to pin 14, and ground is assigned to pin 7. In many 16-pin DIPs, pin 16 is power, and pin 8 is ground. On other 16-pin DIPs power and ground may be on pin 5 and pin 12 respectively. These assignments are typical, **but never assume anything about pin-outs: always use a manufacturer's data manual, device data sheet or some other reliable source.**

Now that you have identified power and ground, do not confuse them. Don't attach a +5 volt power source to the ground pin, or vice versa. Otherwise, goodbye chip.

When hooking up the chip in a circuit, do attach the power and ground first. This procedure avoids two other potential mishaps. You avoid the confusion of power or ground with one of the signal pins, which is just as bad as crossing power and ground themselves. You also avoid the problem of inputting signals to the chip without a power or ground connection. If you input signals without proper power or grounding, you can destroy the IC very quickly.

By identifying these two pins and attaching them correctly to the power and ground lines from the power supply (in this case pins 1 and 8 on the game socket), you will avoid these mixups.

Input Pins. Often, you can connect TTL inputs directly to power (+5 volts). If in doubt, use a current limiting resistor, say in the range of 220 ohms to 1,000 ohms. A 1 K Ω resistor limits the current flowing into the input to about 5 milliamps (mA) or less, which is quite acceptable (Fig. 3-4B).

Why would you want to put +5 volts on a TTL input anyway? Well, you'll occasionally want to *tie-up* one or more inputs in device or circuit to a logic high/binary 1. This is a way of dealing with unused inputs to devices that are not needed on a mul-

device package. Doing this minimizes noise from floating, untied lines and improves circuit performance. Tying an input to logic 1 through such a *pull-up*, current limiting resistor may also be desirable from the standpoint of circuit logic function, as well.

For similar reasons, you may want to put a low or binary zero on an input line, as in Fig. 3-4C. In this case, it is usually safe to make the connection without a resistor, because no current is being drawn from the high side of the power supply. However, in the case of Experiment 1, we used a resistor to *pull down* the pushbutton TTL inputs to low for a good reason. The open and closed Apple function keys in Apple IIes are connected directly

to PB0 and PB1. If one were to press either of these keys with the PB inputs directly grounded, the Apple power supply would reset, just as if the computer were turned off and then on again. Inserting the pull-down resistor limits the excessive current flow that causes such power supply action; this is indicated by the dashed resistor in the figure. Inputs to external TTL devices may be tied to ground directly because they will not, and should not, have more than one line going to them. The only reason that Apple made this dual connection from keyboard and game socket on IIe versions was for the convenience of game players, and for those programmers who might need extra function keys.

Output Pins. These precautions are easy. In

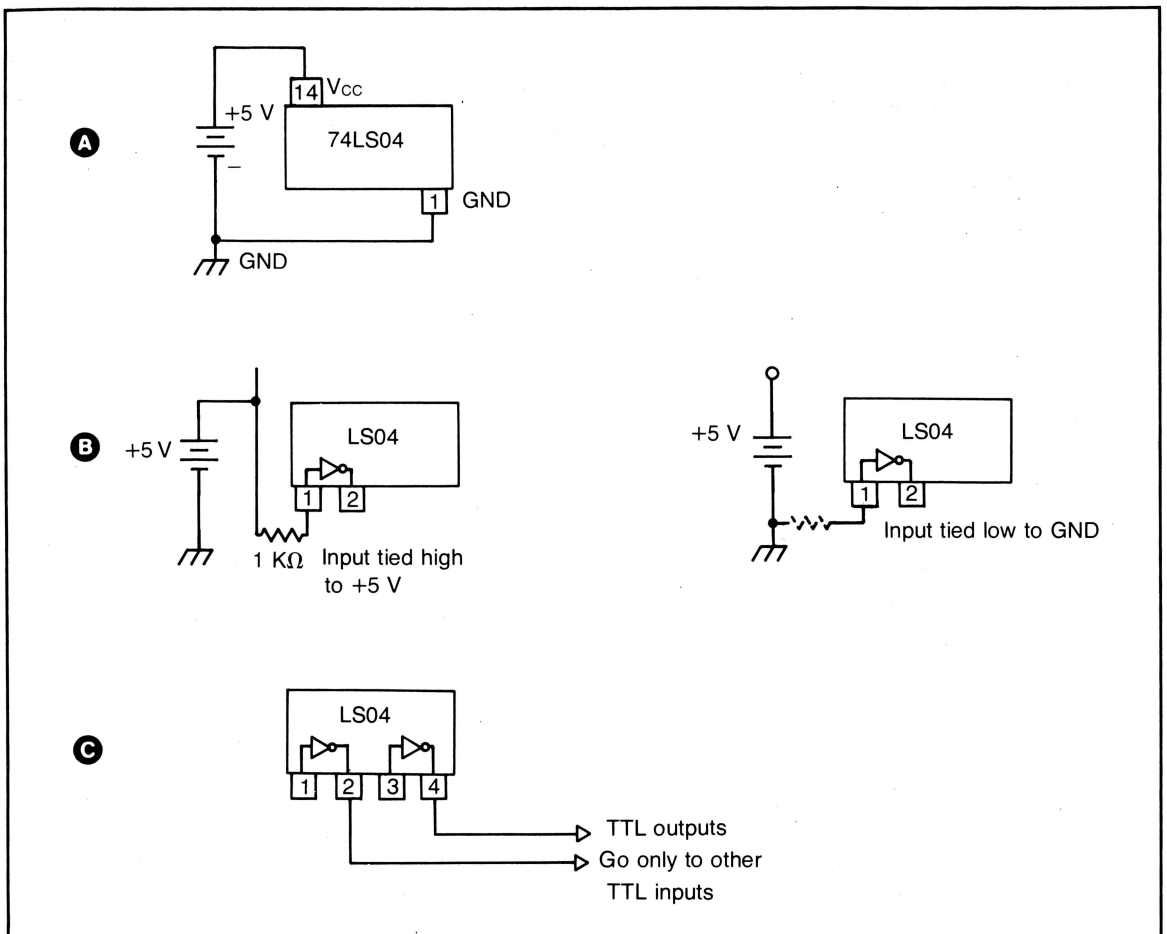


Fig. 3-4. TTL handling precautions. Refer to the text.

fact there are only two. **Don't connect TTL outputs to power or ground, with or without a resistor. Ever.** You will normally attach TTL outputs only to other TTL inputs. The exceptions include interfacing between TTL and electrically dissimilar families, and output loading for purposes of standards testing and quality control. These applications do not concern us.

The other precaution is, **don't connect two TTL outputs together.** In the so-called *totem pole* outputs of the devices we will be using, doing so could do damage to the device.

The TTL annunciator outputs will only connect to the low power Schottky inputs that you will be using in breadboarded circuits. **Never connect the annunciator lines together, to power, or to ground.**

EXPERIMENT 2, NOT AND IS

The function performed by the inverter of the last section is the Boolean NOT function. Little more needs to be said, except to note that it has an alternate name, the inverting buffer. The buffer part of the schematic symbol in Fig. 3-5A is the triangle lying on its side. The inverter symbol is the little circle on the output end of the device. Remember that any time you see one of these circles, either on the input or output side of a device, it means that the associated signal is inverted. That is, a high becomes low and vice versa.

The Boolean IS function is performed by

a noninverting buffer. The symbol for it is the triangle without the circle. Just as in English where a double negative means the same thing as an affirmative statement, a double NOT in digital logic means IS. In Fig. 3-5B two of the inverters from the hex package are placed in series (NOT-NOT) forming the equivalent of a noninverting buffer (IS).

Inverting buffers are often used for their negating logical properties. At other times, they are used to reverse the polarity of a signal for purely electrical tasks such as turning a transistor on or off so that it acts like a switch.

The utility of a noninverting buffer may seem dubious, at least for performing a logical function. This is true, but there are cases where a weak digital signal must first be boosted up to a normal TTL output level, so that it can effectively drive other TTL or even transistor inputs. Buffers provide this electrical function, because they can normalize the input signal to a standard output drive or power level. Buffers of both varieties can also be specially designed to have high output power levels so that they can drive signals down long transmission lines, which tend to dissipate power. Again, this is more of an electrical function than a logical function, but a necessary one if you are going to connect TTL circuits to the outside world.

Purpose

To examine inverting and noninverting buffers—the Boolean NOT and IS functions.

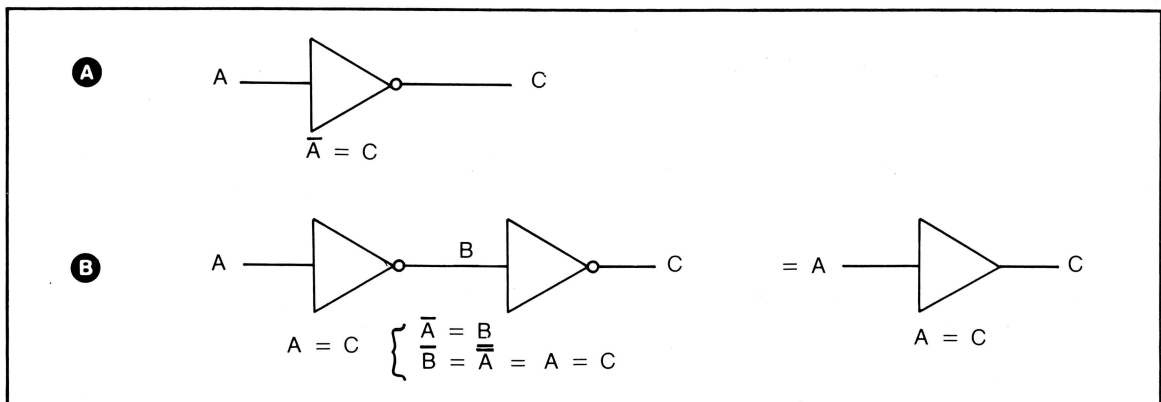


Fig. 3-5. Schematic logic symbols for NOT and IS.

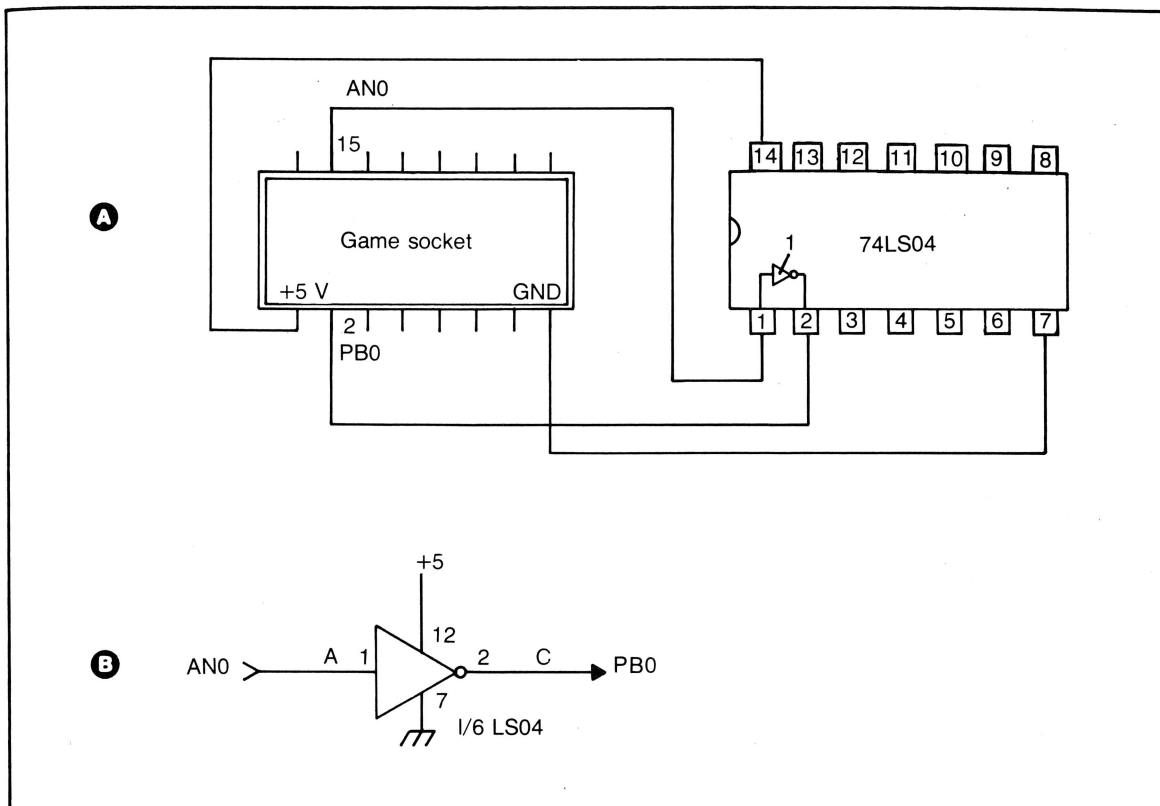


Fig. 3-6. Setup for Experiment 2, NOT.

Materials

1 - 74LS04 Hex Inverter Package

Procedure

1. With the computer off and the game jumper plugged into one end of the breadboard, wire up the circuit illustrated in Fig. 3-6. Use either the semi-schematic representation of Fig. 3-5A or the more conventional schematic form of Fig. 3-6B as a guide. The latter is the one we will soon be using exclusively. Get into the habit of attaching power and ground, pin 14 and pin 7 on this chip, before attaching the signal lines, pin 1 and pin 2. Leave PB0 unconnected until you turn on the computer.

2. Turn the computer on, attach PB0 and run BDIS. Input the following sequence to reconfigure the display:

Ctrl-R / A / 1,1 / C.

You'll enter the reconfigurations mode first, select choice A to change the column format, choose a 1 input/1 output format, and return to the main program with C.

3. Generate the 2-line truth table using only the 0 key on the keyboard to toggle annunciator zero (AN0). The result should be the same as in Table 3-1A.

This is the same as that in Fig. 3-3. (Suggestion: Measure the output voltage for each of the two input states with your voltmeter. Put ground lead to pin 8 on the game socket, and put test lead to pin 2 on the IC.)

4. To demonstrate the noninverting buffer, hook up the circuit shown in Fig. 3-7. Use either A or B

Table 3-1. Experiment 2, NOT and IS Functions.

A

GPSIG: AN0: PB0		
GPIN#:	15:	2
LABL1:	A :	W
LABL2:	0 :	0

0	0 :	1
1	1 :	0

B

GPSIG: AN0: PB0		
GPIN#:	15:	2
LABL1:	A :	W
LABL2:	0 :	0

0	0 :	0
1	1 :	1

as a guide. You can make the proper connections without turning the computer off because you need not touch power or ground. Simply disconnect the PB0 lead from pin 2 and reinsert it into one of the socket holes in line with pin 4. Then place a short jumper wire between pin 2 (inverter 1s output) to pin 3 (inverter 2s input).

5. Clear the screen with Ctrl-H and generate another 2-line truth table. Check your results against Table 3-1B.

Discussion

Now that you have hooked up a circuit and actually demonstrated the operation of a digital IC, you should be more comfortable using the computer-based logic trainer. The hex inverter was chosen not only for its simplicity, but also because the idea of inversion and noninversion is important for understanding the next four logic devices. If you

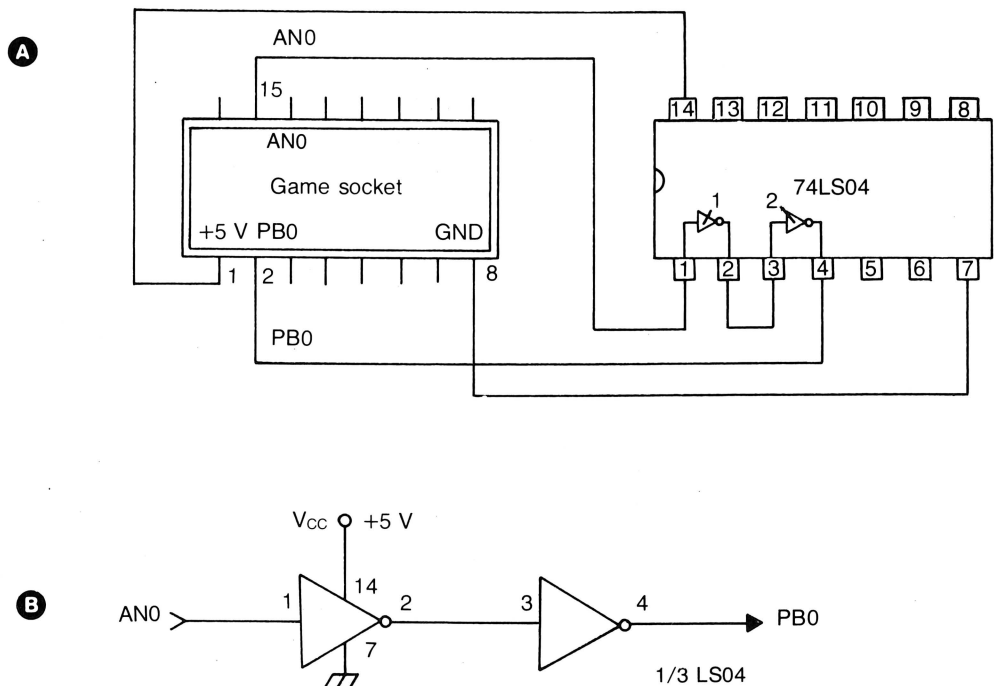
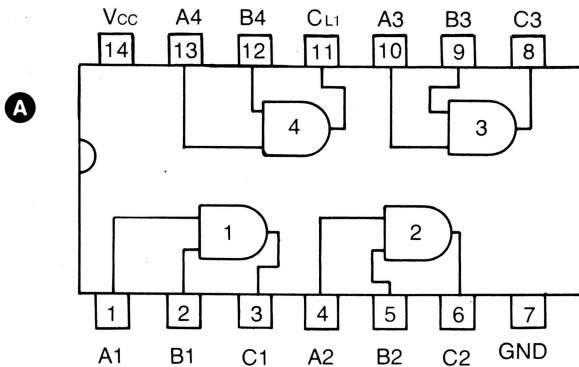
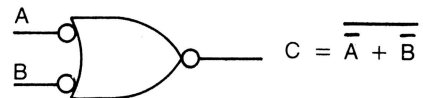
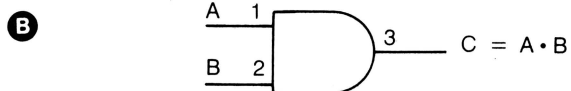


Fig. 3-7. Setup for Experiment 2, IS.



74LS08 Quadraple 2-input AND

$$A \cdot B = C$$



C

Line #	Inputs		Output
	B	A	C
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

OR-form complementary definition

AND-form Primary definition

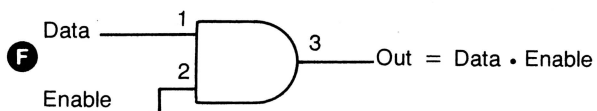
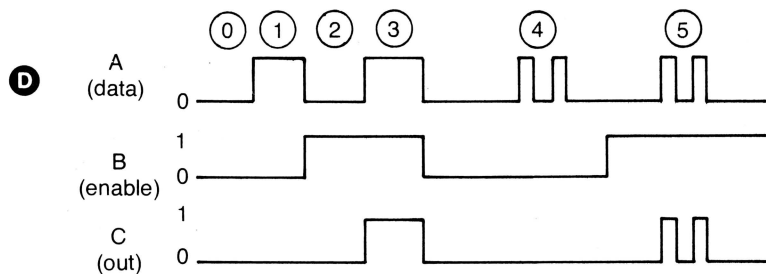


Fig. 3-8. Description of AND. Refer to the text.

measured the output voltage levels, you should have obtained a 3.0 to 4.0 volt range for the TTL logic high/binary 1, and a value near 0 volts for the logic low/binary 0 output.

EXPERIMENT 3, AND

The primary AND-form definition for the AND function is:

The output of AND is high/binary 1 if and only if **all** of the inputs are high.

The complementary OR-form definition is:

The output of AND is low/binary 0 if any **one** of the inputs is low.

A typical AND IC is the 74LS08 package. The pin-out of this IC is shown in Fig. 3-8A. One of the AND devices has been isolated and drawn in Fig. 3-8B. The schematic symbol to the left is accompanied by its corresponding equation. $A \cdot B = C$. Both the equation and the symbol correlate with the primary definition for AND just given.

Note that the keyword **all**, the dot in the equation, and the AND device schematic symbol mean the same thing; if all the inputs have the same state (two inputs high in this case), then the output will be thus and such (logic high for an AND device). Now look at the truth table in Fig. 3-8C. Note that all, in this case the two, inputs have the same state, high/binary 1, only in line 3. When this is the case, and only when this is the case, will the output be high. This is what is meant by the AND-form of device description.

We can also describe the AND function in a different way, as a sort of modified OR device!

To the right in Fig. 3-8B is the alternative representation: the OR schematic symbol with both its inputs and its output inverted. The associated equation, $\overline{A} + \overline{B} = \overline{C}$, says that when either (plus sign means either-or) of the inputs are low/binary 0 (a bar over each letter and a circle at each input), then the output will be low. A bar over the whole input term means that the net result is a low output, and corresponds the circle on the output end of the

symbol. Look at the truth table again. In the first three lines, 0-2, one of the inputs is low/binary 0. And if, as in the complementary definition, any one of the inputs is low, then the output will also be low (OR-form).

These two ways of looking at the AND device, with two definitions, two symbols and two equations, together constitute what is known as *gate equivalence*. That is, there is an *AND-form* and an *OR-form* of device representation. This may be an odd concept to grasp, so study these relationships carefully, again and again if necessary. Only go on after this material is clear. There is a bonus for the hard work: the other three devices which you will study follow the same pattern. So if you understand the AND function, you've got a good jump on the other functions already.

Now look at Fig. 3-8D, the timing diagram. It was designed to illustrate the functions of AND, and to complement the truth table, the definitions, and indirectly, the logic equations. The first four points, 0-3, define all four possible input states of a two-input device ($2^2 = 4$), and correspond to lines 0-3 in the truth table. Point 3 corresponds to the general definition of AND. Points 0-2 correspond to the complementary definition of AND. Carefully compare these points in the timing diagram with their respective definitions to see that this is so.

Implied in both the truth table and the timing diagram is the property of an AND device as a noninverting buffer. If you tied both inputs together, as in Fig. 3-8E, and fed one digital signal to both, then they would have the same logic level; high or low. That is, there would only be two possible states for the device: points 0 and 3 in the timing diagram and lines 0 and 3 in the truth table. As both representations show, the output signal is the same as the input signal, or noninverted. The equation and symbol for this input/output relationship is given in Fig. 3-8E, and is the same as that studied in Experiment 2.

There is one other property of AND which is clearly shown in the timing diagram at points 4 and 5: as a noninverting gate with an active high enable.

This idea of an *enabling* or *gating* function is easier to see if we relabel the inputs A and B as data

and enable respectively and the output simply as OUT. See Fig. 3-8F. If the enable line is low/0, as at point 4 on the timing diagram, then the two data pulses cannot get through to the output, which remains at logic low. If the device is *activated* by placing a high on enable (binary 1), as at point 5, then the device will pass the two data pulses to the output. This action is much like opening or closing a gate, hence the term. Because the enable line must be high (see lines 2 and 3 in the truth table) for the

data to be passed to the output, one refers to the line as an *active high* enable.

Purpose

To demonstrate the LS08 device and its properties as a logic element, noninverting buffer, and noninverting gate.

Materials

1 - 74LS08 Quadruple 2-input AND IC package

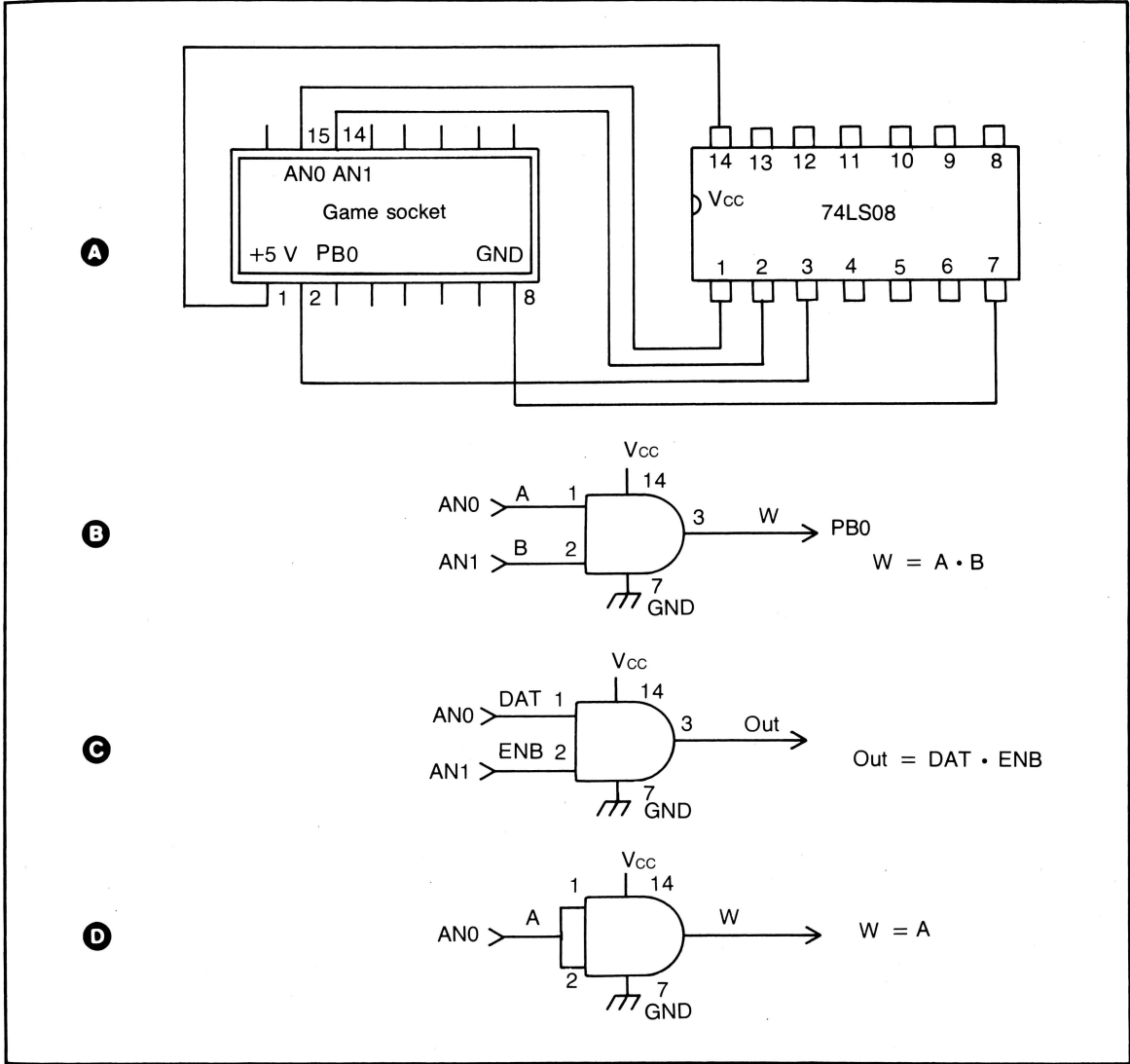


Fig. 3-9. Setup for Experiment 3, AND.

Procedure

1. With the computer off, hook up the breadboard circuit using Fig. 3-9A or B as a guide. This configuration is used to study the AND function; make sure the connections are correct before turning on the computer. Leave PB0 unconnected.

2. Turn on the computer, attach PB0, and then run BDIS. Configure it in a 2 input/1 output format with the sequence.

Ctrl-C / A / 2,1 / C.

3. Demonstrate the AND function by generating the 4-line truth table. Note that you are counting up in binary—00, 01, 10, 11—if you enter the inputs as indicated in Table 3-2A.

4. To show the enabling function, it is desirable to reconfigure the screen. The circuit is identical to the AND device just examined, but the signal labels should be changed to those of Fig. 3-9C. Reconfigure with the following sequence:

Ctrl-R / B / 2 / "ENBDAT OUT" / Y / A / 2,1 / C.

5. As usual, you will return to the main program with the annunciators at logic low. Toggle the data line on and off. You should see no effect on the output line, which remains at 0, just as in point 4 in the timing diagram. The gate is in effect closed to signals from the data line.

6. Generate the second entry line with return. Toggle ENB to on (binary 1). Now, as you toggle data on and off, you will see the signal appear on the Out line in noninverted form. This is the same as point 5 on the timing diagram. Your results should be similar to Table 3-2B (the 0/1 in the table indicates alternating states or toggling of the line).

7. Finally, demonstrate the noninverting buffer function of AND. First remove the AN1 line from pin 2. Then place a short jumper from pin 1 to pin 2, as in Fig. 3-9D. (Remember, never have two TTL outputs connected together. If you hadn't removed

the annunciator line AN1 before jumping to two AND inputs together, you would have been tying AN0 and AN1 together.)

Now change the screen display into a 1 input/1 output format with the old default labels. The quickest way to do this is to quit BDIS with Ctrl-Q and then rerun it. Enter

Ctrl-R / A / 1,1 / C.

8. Generate a 2-line truth table for the buffer. You should obtain a result similar to that of Table 3-2C, the same as for the IS function demonstrated earlier.

Table 3-2. Experiment 3, AND Function.

GPSIG: AN1 AN0: PB0			
GPIN#:	14	15:	2
LABL1:	B	A :	W
LABL2:	1	0 :	0

0	0	0 :	0
1	0	1 :	0
2	1	0 :	0
3	1	1 :	1
GPSIG: AN1 AN0: PB0			
GPIN#:	14	15:	2
LABL1:	B	A :	W
LABL2:	ENB	DAT:	OUT

0	0	0/1 :	0
1	1	0/1 :	0/1
GPSIG: AN0: PB0			
GPIN#:	15:	2	
LABL1:	A :	W	
LABL2:	0 :	0	

0	0 :	0	
1	1 :	1	

Discussion

You have seen how an AND device can function as a logic element, as a noninverting gate for data flow with active high enable, and as a noninverting buffer. This distinction between a gate and a logic element is actually a matter of intended use. The term logic element might apply to part of an arithmetic or control circuit, while the term gate might apply to a circuit element where one desires to enable/disable signal passage at will.

Sometimes a combinational logic device will be referred to as a gate. This does not imply the enabling function, but it is rather a generic term which usually denotes an SSI package or device that performs the AND, OR, NAND, or NOR function.

The experiment also underscored the AND-form and OR-form of device description: **all** inputs 1 - then output is 1. **Any** input 0 - then output is 0.

If you consult a TTL data manual you will see that a number of AND packages have three and even four input AND devices, the LS11 and LS21 for example. Everything said about the 2-input AND device applies to these as well.

EXPERIMENT 4, NAND

The primary, AND-form definition for the NAND function is:

The output of NAND will be low/binary 0 if and only if **all** of the inputs are high/binary 1.

The complementary, OR-form definition for NAND is:

The output of NAND will be high/binary 1 if **any one** of the inputs is low/binary 0.

If you compare these definitions with those for the AND function, you will find that the output states, high or low, have been reversed, just as you would expect for the inverted output. But the format of the definitions is otherwise similar, with the primary definition having an AND-form and the complementary definition an OR-form. Each form of definition has its corresponding logic symbol and

associated equation, shown in Fig. 3-10B with the primary AND-form to the left and complementary OR-form to the right. In this case, the NAND gate is the *equivalent* of an OR gate with negated inputs.

Referring to the truth table you can see that line 3, where all (both/and) inputs are high/1, the output is low. In lines 0-2, where only one of the inputs is low, the output is high. These two situations are nothing more than restatements of the primary and complementary definitions, as with the AND function.

Examine the timing diagram, Fig. 3-10D. Note the correspondence among points 0-3, lines 0-3 in the truth table, and both definitions for the NAND function. Line 3 and point 3 relate to the primary definition, and lines and points 0-2 to the complementary definition. The timing diagram is simply another way of representing the logic function.

Figure 3-10A illustrates that the pin-out for the LS00 NAND package is the same as that for the LS08 AND IC. Power, ground, input and output pins are identical.

NAND acts as an inverting buffer if the inputs are tied together, as in Fig. 3-10E. This operation is implied in lines 0 and 3 in the truth table.

Finally, NAND acts as an inverting gate with active high enable. The arrangement in Fig. 3-10F is the same as in Fig. 3-10B with the appropriate labels. The output will change with data in inverted form only when enable is high. Point 5 on the timing diagram illustrates this fact. It will remain low and unresponsive if enable is low, as at point 4.

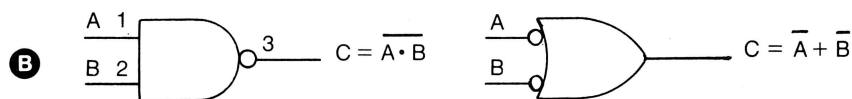
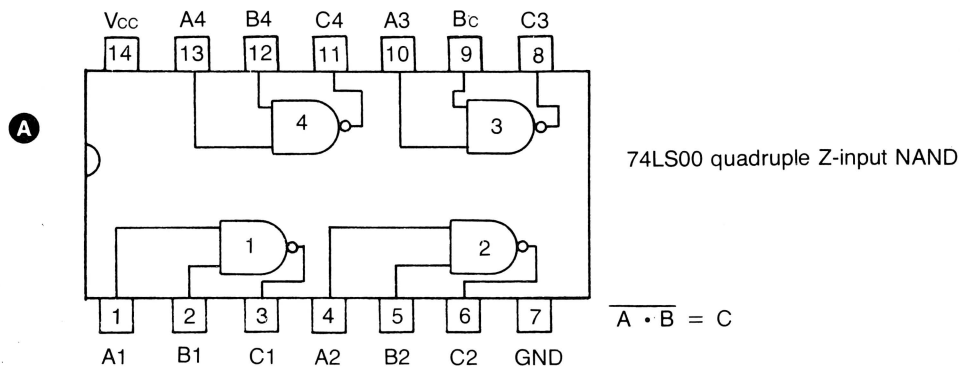
Just as in Fig. 3-8, there is a great deal of information packed into Fig. 3-10. Fortunately, the overall pattern is similar. Let's reinforce the above concepts with a demonstration of NAND properties.

Purpose

To demonstrate the function of a NAND device as a logic element, an inverting buffer, and as an inverting gate with active high enable.

Materials

1 - 74LS00 Quadruple 2-input NAND package



C

Line #	Inputs		Output
	B	A	C
0	0	0	1
1	0	1	1
2	1	0	1
3	1	1	0

OR-form complementary definition

AND-form primary definition

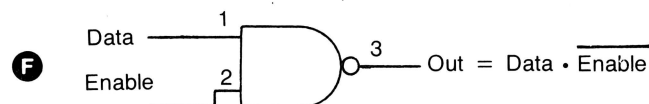
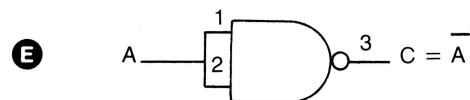
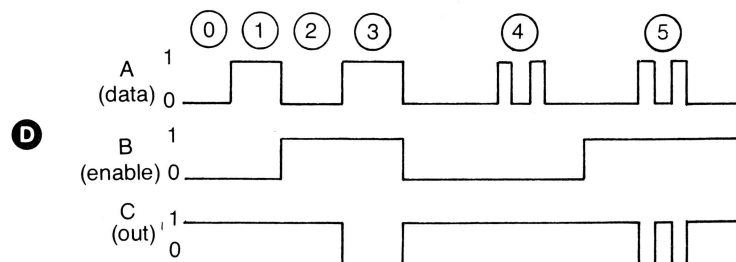


Fig. 3-10. Description of NAND.

Procedure

Notes on changing chips. Before starting there are two alternative suggestions regarding chip and hookup changes between experiments.

To save a little wear and tear on the Apple's power switch, you may want to buy one of the multiple outlet receptacle boxes that has its own on/off switch. These are available in any hardware or department store.

Alternatively, and more preferable, there is a way to change chips between experiments without turning off the computer and having to rerun and

reconfigure BDIS. This is possible if you adhere to the following. First, remove all signal lines—AN and PB—from the chip or chips on your breadboard. Second, remove the +5 volt power line from the power pin on the chip. Third, remove ground from the chip's ground pin. Remove the chip itself from the breadboard.

To install a new chip, insert it on the breadboard and then reverse the above procedure: attach the ground line first, then the +5 volts Vcc line, and then the signal lines. All of this adheres to the caution against applying signals with no power, and

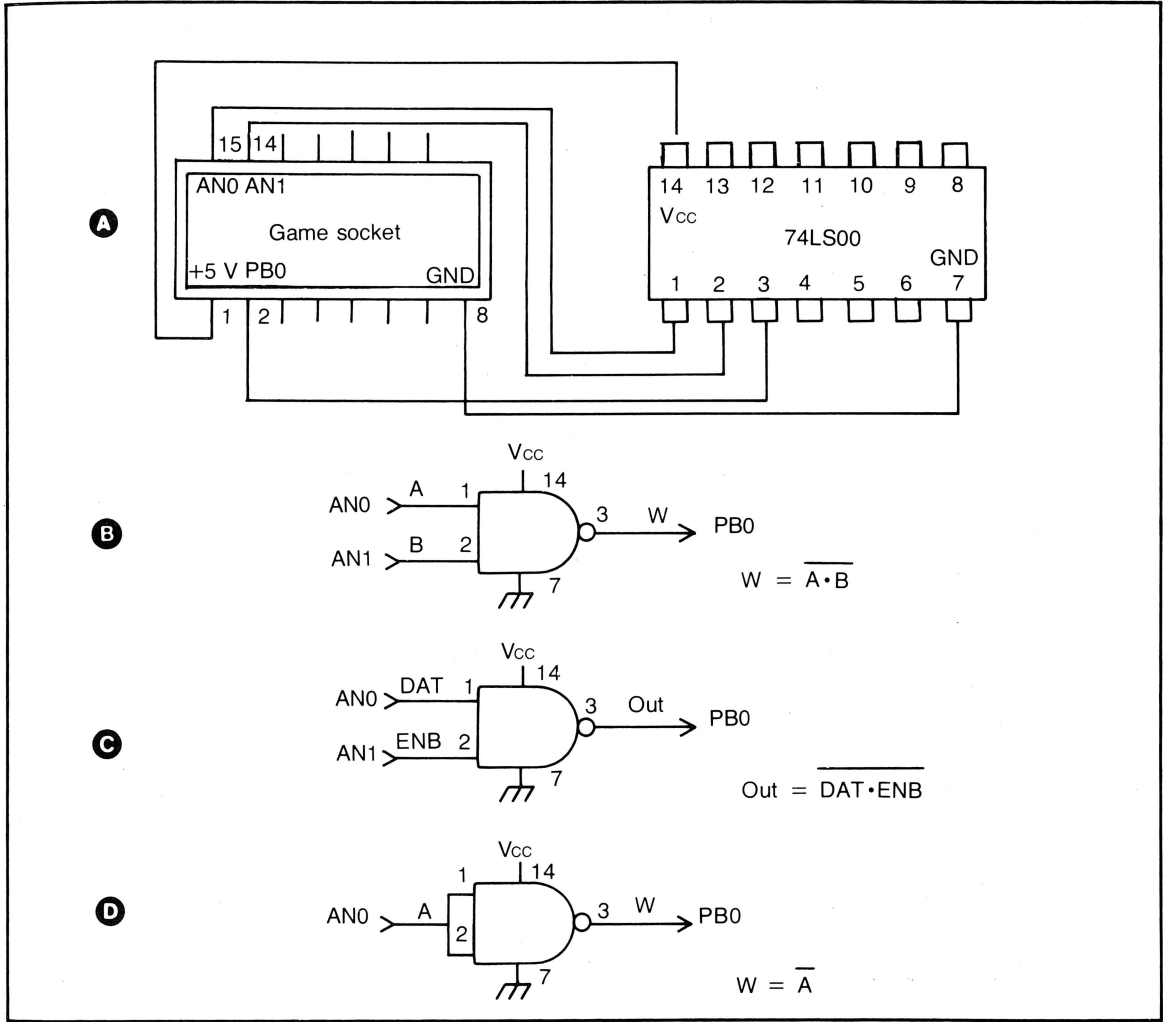


Fig. 3-11. Setup for Experiment 4, NAND.

to the suggestions that ground and power lines be connected first and in that order if the power line is live at the time of connection.

1. With the computer off, connect the circuit using Fig. 3-11A or B. Leave PB0 unconnected to pin 3 of the LS00. Turn on the computer, connect PB0 and run BDIS. Alternatively, you could leave the computer on with BDIS running; just follow the suggestions in the note just presented.

2. Configure the BDIS display to a 2-input/1-output format, if necessary:

Ctrl-R / A / 2,1 / C.

3. Generate a 4-line truth table so that all possible input states are displayed. Compare your results with Table 3-3A.

4. Show how the NAND device acts as an inverting gate with active high enable. Relabel the input and output columns in the screen display as shown earlier in item 4 of the last experiment. See Fig. 3-11C.

5. Try toggling the data line with a low/0 on enable. Then generate the second line, set enable high, and toggle data on and off again. Note the inversion of the data signal. See Table 3-3B.

6. Show how the NAND device acts as an inverting buffer, or inverter. Without touching power or ground, modify the circuit to look like that in Fig. 3-11D. You must remove one of the annunciator lines from its input pin first, say AN1 from pin 2, before connecting the two inputs together. You do this to avoid connecting two TTL outputs together, as mentioned earlier. Now you can relabel the headings as before.

Ctrl-Q / RUN / Ctrl-R / A / 2,1 / C.

7. Generate the 2-line table of an inverting buffer; it should look like Table 3-3C.

Table 3-3. Experiment 4, NAND Function.

```
GPSIG: AN1 AN0: PB0
GPIN#:  14  15:  2
LABEL1:  B   A :  W
LABEL2:  1   0 :  0
```

```
-----
0      0      0 :  1
1      0      1 :  1
2      1      0 :  1
3      1      1 :  0
```

A

```
GPSIG: AN1 AN0: PB0
GPIN#:  14  15:  2
LABEL1:  B   A :  W
LABEL2: ENB DAT: OUT
```

```
-----
0      0 0/1 :  1
1      1 0/1 : 1/0
```

B

```
GPSIG: AN0: PB0
GPIN#:  15:  2
LABEL1:  A :  W
LABEL2:  0 :  0
```

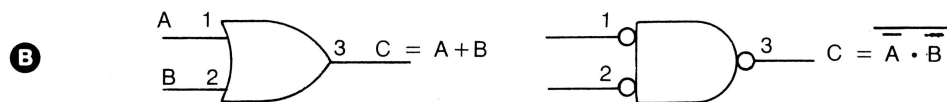
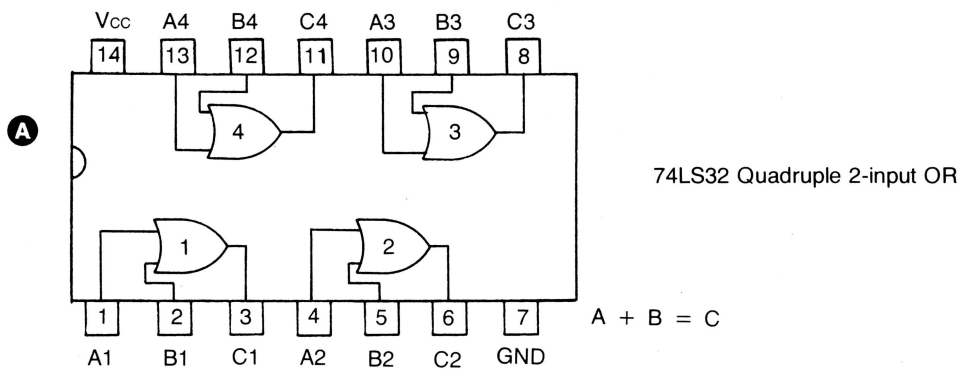
```
-----
0      0 :  1
1      1 :  0
```

C

Summary

Buffer, gate and logic element functions of the NAND device have been examined. Some of this probably seems like hard work. But you are learning the fundamentals about the functional aspects of combinational SSI devices.

As an exercise, consult a TTL data book and find NAND packages on which the devices have more than two inputs. The 74LS10 NANDs have



C

Line #	Inputs		Output
	B	A	C
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

AND-form complementary definition

OR-form primary definition

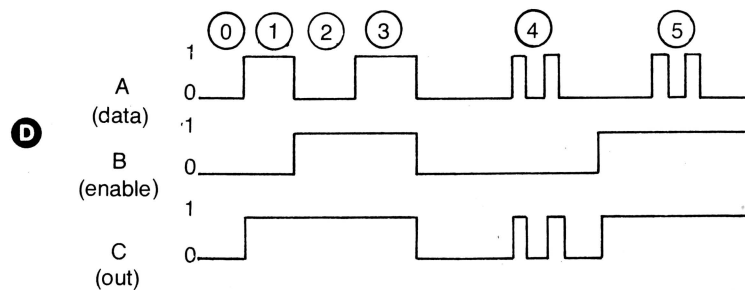


Fig. 3-12. Description of OR.

three inputs per device; there are others which have four and even eight.

EXPERIMENT 5, OR

The primary, OR-form definition for the OR function is:

The output of OR will be high/1 if any **one** (or more) of the inputs is high.

The complementary, AND-form definition for OR is:

The output will be low/0 if and only if **all** (each and every) of the inputs are low.

Figure 3-12A is the pin-out of the 74LS32 Quad OR package. The pin assignments for input, output, power and ground is the same as those for the LS00 NAND and the LS08 AND packages.

The OR symbol and equation which relates to the primary definition is to the left in Fig. 3-12B, the OR-form of representation. To the right is the alternative or equivalent symbol and equation that correspond to the complementary AND-form. The gate equivalence principle states that an AND device with both the inputs and the output inverted (such as by LS04 devices) will perform the OR function.

As before, the truth table spells out each of the four possible states of this 2-input device. Figure 3-12C. A timing diagram can be constructed to reflect the four states, as indicated at points 0-3 in Fig. 3-12D. Note the difference between this table and those for AND and NAND. This time it is lines 1-3 and points 1-3 which relate to the primary definition, while line and point 0 relates to the complementary definition. This is because the OR-form of device representation is, as expected, the primary means of description.

The OR device becomes a noninverting buffer if you input a signal common to both input lines, as in Fig. 3-12E.

The OR device can act as a noninverting gate with active low enable, shown in Fig. 3-12F. Unlike the AND and NAND gates, the enabling signal must be low/0 in order for data to pass to the

output, because the output can change states (0/1) only when a low is placed on the enable line. See points 0 and 1 in the timing diagram and lines 0 and 1 in the truth table. Again, points 4 and 5 were included in the timing diagram to graphically illustrate this gating effect.

Purpose

To demonstrate the operation of an OR device as a logic element, as a noninverting buffer and as a noninverting gate with active low enable.

Materials

1 - 74LS32 Quadraple 2-input OR package.

Procedure

1. With the computer off, wire up the circuit shown in the breadboard semi-schematic or in the more conventional schematic diagram of Fig. 3-13 or follow the steps for changing chips given in Experiment 4.
2. If necessary, configure the display in a 2-input/1-output format.
3. Generate the 4-line truth table for OR as a logic element, and compare with Table 3-4A. Remember, you can print to screen at any time with Ctrl-P.
4. Relabel the display to read "EN' DAT OUT" in 2 in/1 out format. The use of the apostrophe in place of the bar indicates an active low enable.
5. Now demonstrate the active low enable, noninverting gate function of this OR device. Again, the device is still performing the OR function. No circuit changes have been made. It is only the intention of the use reflected by the reassignment of new signal labels which justifies the term noninverting gate.
6. Finally, remove the annunciator AN1 lead from pin 2, and then tie the two input pins together with a short jumper. Reconfigure the display for 2 in/1 out

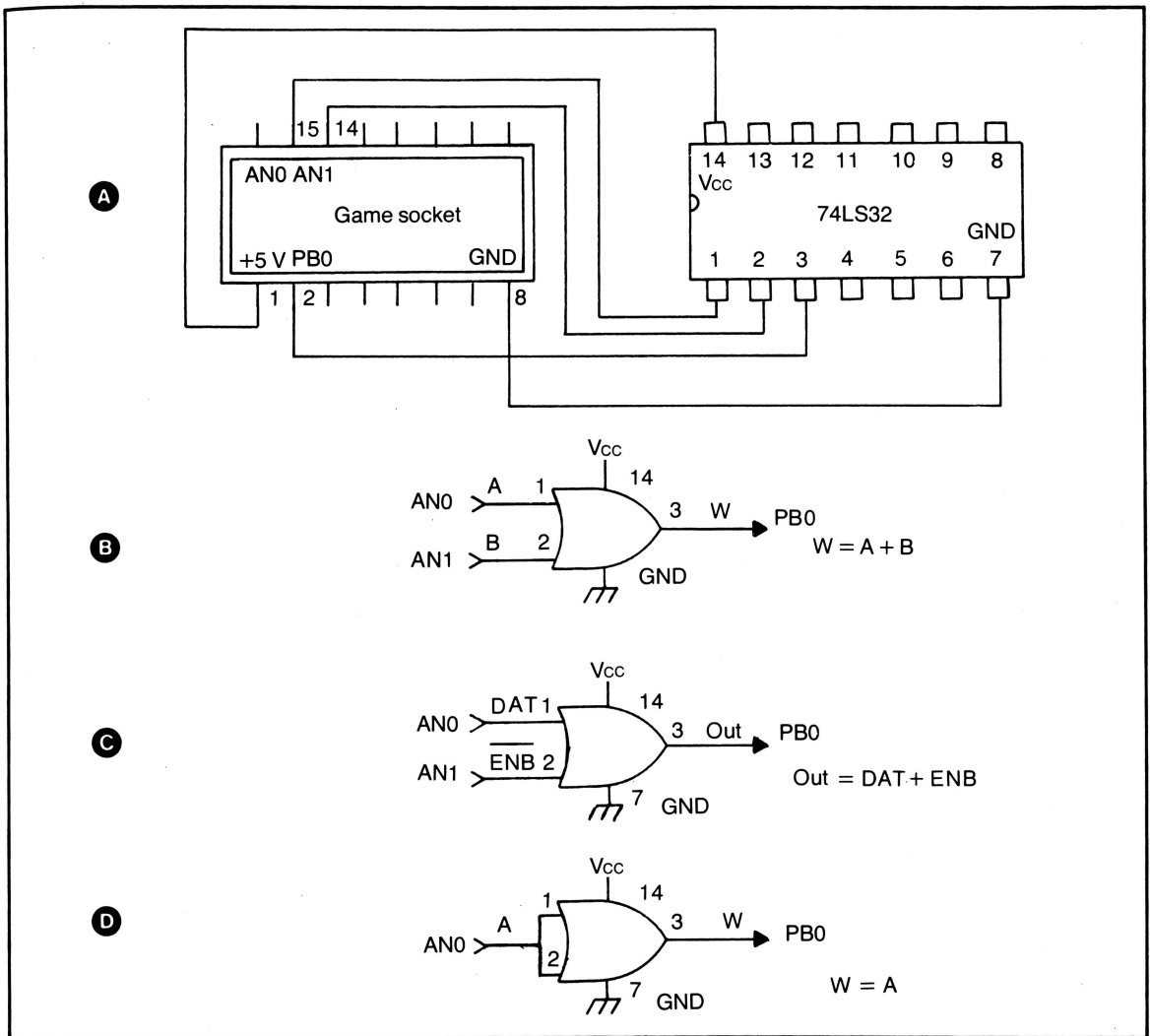


Fig. 3-13. Setup for Experiment 5, OR.

and generate the 2-line truth table for a noninverting buffer.

EXPERIMENT 6, NOR

The primary, OR-form definition for NOR is:

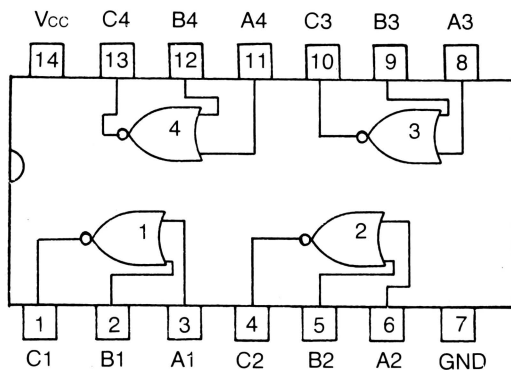
The output of NOR will be low/0 if any **one** (or more) of the inputs is high/1.

The complementary, AND-form definition for the NOR function is:

The output of NOR will be high/1 if and only if **all** of the inputs are low/0.

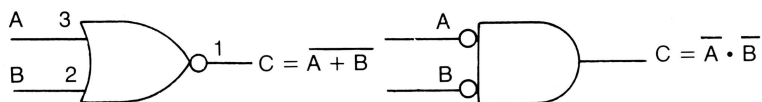
If you compare these definitions with those for the OR function, you'll see that the output states are reversed: high for low and vice versa, just as expected for an inverted output.

As with the OR function, there is a parallel among the primary definition, the primary schematic symbol and associated equation in Fig. 3-14B (left), and lines 1-3 in the truth table.

A

74LS02 Quad 2- input NOR

$$\overline{A + B} = C$$

B**C**

Line #	Inputs		Output
	B	A	C
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

AND-form complementary definition

OR-form primary definition

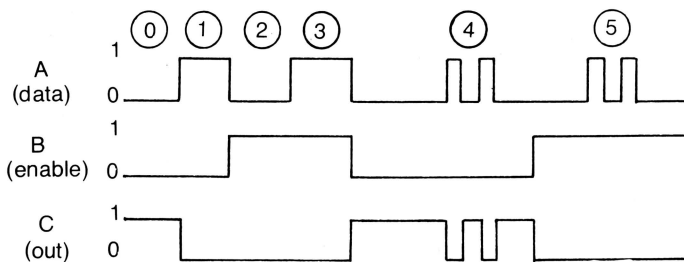
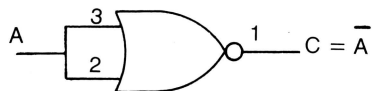
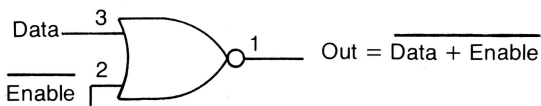
D**E****F**

Fig. 3-14. Description of NOR.

Table 3-4. Experiment 5, OR Function.

GPSIG: AN1 AN0: PB0	GPSIG: AN1 AN0: PB0	GPSIG: AN0: PB0
GPIN#: 14 15: 2	GPIN#: 14 15: 2	GPIN#: 15: 2
LABL1: B A : W	LABL1: B A : W	LABL1: A : W
LABL2: 1 0 : 0	LABL2: EN' DAT: OUT	LABL2: 0 : 0
<hr/>		
0 0 0 : 0	0 0 1/0 : 1/0	0 0 : 0
1 0 1 : 1	1 1 1/0 : 1	1 1 : 1
2 1 0 : 1		
3 1 1 : 1		
(A)	(B)	(C)

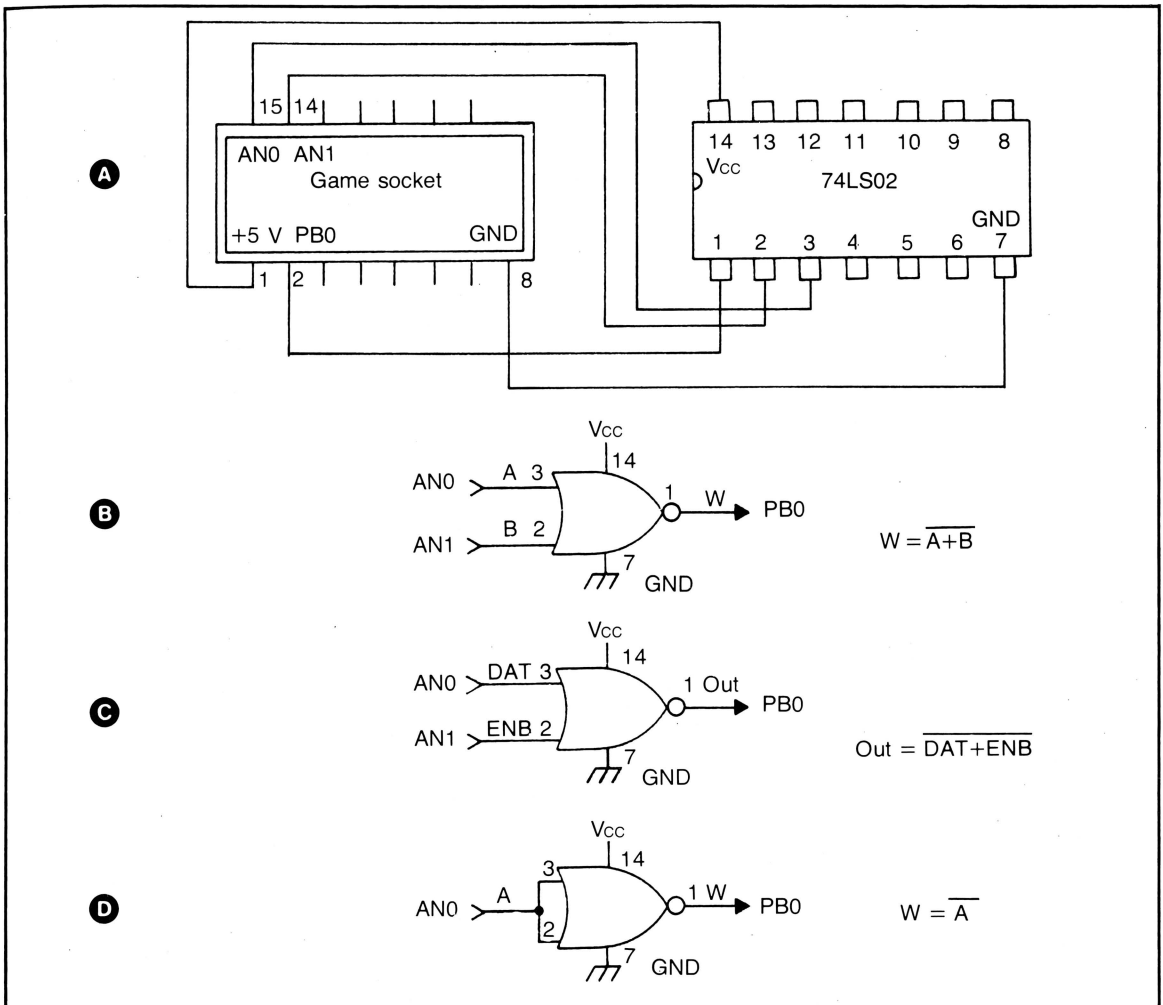


Fig. 3-15. Setup for Experiment 6, NOR.

The timing diagram serves as a graphic aid as before.

Gate equivalence also applies to the NOR device. It states that an AND device with inverted inputs performs the NOR function, as shown in Fig. 3-14B.

Figure 3-14A shows the pin-out of a quad 2-input NOR package, the 74LS32. Note particularly that the order of the input and output pins on this 14-pin package are reversed in comparison with the other three gate packages. However, power and ground are still on pins 14 and 7, respectively.

NOR devices can act as inverting buffers, or inverters, must like NAND. By tying the two inputs together (Fig. 3-14E), you have the NOT function.

Last, NOR devices can act as inverting gates with an active low enable (Fig. 3-14F). The active low requirement for the enabling line is indicated by a bar over the label. Points 4 and 5 on the timing diagram illustrate this passage or nonpassage of data to the output.

Purpose

To examine a typical NOR device as a logic element performing the NOR function, as an inverter, and as an inverting gate with active low enable.

Materials

1 - 74LS02 Quadruple 2-input NOR package

Procedure

1. You can follow either procedure: Computer off. Use Fig. 3-15A or B as circuit guides for the NOR demonstration. Computer on. Run BDIS, configure a 2 in/1 out column format and generate a truth table for NOR (Table 3-15A).

2. Or change chips by removing signals, then +5 V power, then GND. Then install the LS02 and reverse the procedure.

3. Relabel the display headings using "EN' DAT OUT" as you did with NAND, if necessary. Generate the table for an inverting gate with active low enable (Table 3-15B).

Table 3-5. Experiment 6, NOR Function.

GPSIG:	AN1	AN0:	PB0
GPIN#:	14	15:	2
LABL1:	B	A:	W
LABL2:	1	0:	0

0	0	0 :	1
1	0	1 :	0
2	1	0 :	0
3	1	1 :	0

A

GPSIG:	AN1	AN0:	PB0
GPIN#:	14	15:	2
LABL1:	B	A:	W
LABL2:	EN'	DAT:	OUT

0	0	1/0 :	0/1
1	1	1/0 :	0

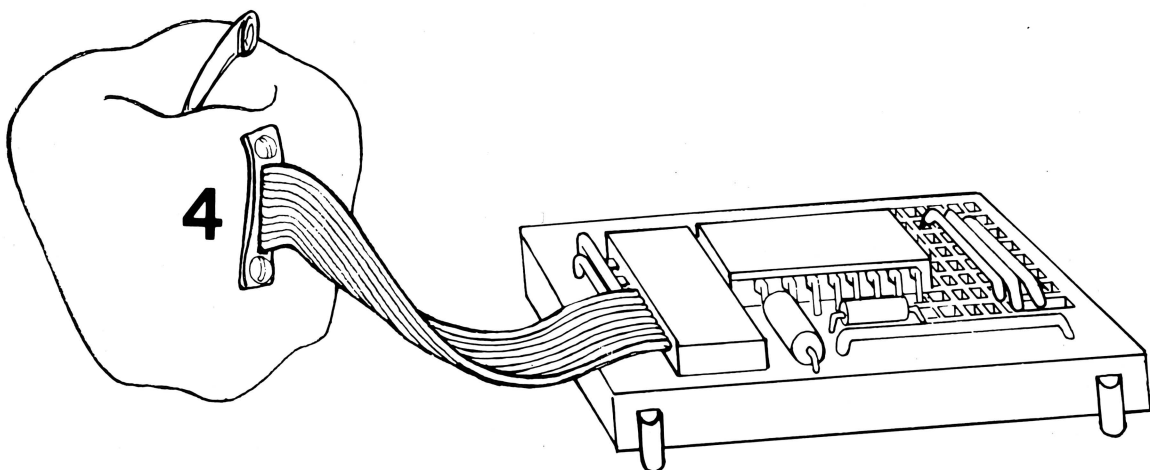
B

GPSIG:	AN0:	PB0
GPIN#:	15:	2
LABL1:	A:	W
LABL2:	0:	0

0	0 :	1
1	1 :	0

C

4. Remove AN1 from pin 1 first. Tie the inputs together with a short wire. Relabel the signals on the display and generate the 2-line table for an inverter (Fig. 3-15C).



Boolean Methods

Boolean algebra is a simple but powerful notational method for expressing logical relationships. Through the use of Boolean logic equations, truth tables and schematic circuit diagrams, you will be able to analyze and understand logic circuits and design new ones to your own specifications.

The goals of this chapter are well defined and modest. The emphasis is on symbolic methods involving combinational logic, because such methods are very accessible to the beginner and also the most useful in practical situations. Symbolic manipulation of sequential logic relationships is an advanced topic.

Obviously, one chapter's worth of material is not going to turn you into a digital design engineer. But you will acquire the skills to freely translate from truth table to logic equation to circuit schematic, and back again. You will be able to simplify logic equations and thereby minimize circuit complexity. You will also see how to make one device do the work of several. Finally, you will have a clear idea of the how's and why's of combinational circuit

operation from the functional standpoint, and be able to design simple SSI circuits from scratch.

THE RULES OF BOOLEAN ALGEBRA

Some definitions and concepts are necessary before you can begin using Boolean methods for circuit analysis and design. First there is the equation itself, which consists of Boolean *variables* that are related to one another by the Boolean *operators*—AND, OR and NOT—in an *expression*. This expression, which may have one or more subdivisions called *terms*, is equated to a single variable. For example, the elements just mentioned can be identified in Equation 4-1.

$$(A + \bar{B}) \cdot C = X \quad 4-1$$

Each Boolean variable in the equation is represented by a letter, and can assume either of two values: true/binary 1/voltage high or false/binary 0/voltage low. Boolean operators include AND, OR and NOT, and are represented by a dot (\cdot), plus sign

(+), or bar respectively. The equation above is written with three variables to the left, representing inputs, and a single variable to the right of the equals sign, representing the output variable. Figure 4-1A shows how the input variables are grouped into terms, one containing the variables A and B and the other containing C alone. Since the entire expression to the left can assume only one of two values (high/low, etc.), it is convenient to equate this expression to a single output variable, X.

Figure 4-1B is the schematic form of Equation 4-1. Shown is the step-wise build-up of the entire expression, term by term. Variable C is ANDed with a two-variable term, and this *equal footing* of C with A+B explains why C can be considered as a term, as well as a variable. Note that there is a correspondence between each numbered term in the equation and each numbered gate in the schematic. When writing a literal schematic from a logic equation this is generally the case.

Observe that the equation may also be written as $(A+B)C=X$, as on the output line in the schematic. That is, the dot for AND may be omitted with no change in the sense of the equation.

Whenever you are writing Boolean equations it is very important to group the variables and

operators so that the meaning of the expression is unambiguous. Groups of variables and operators are set off by parentheses to make the meaning clear. In some cases, such as in Equation 4-1, this is not absolutely necessary: the *priority rule* states that the variables are ANDed before they are ORed. This rule is similar just as to conventional algebra where variables are multiplied before they are added. However, this convention has limited application, and it is much safer to group variables by parentheses.

A striking example of this necessity for grouping is given in Equation 4-2 and Fig. 4-2.

$$A + B \cdot C + D = ? \quad 4-2$$

Equation 4-2 is similar to Equation 4-1 except that one more variable, D, has been added, and parentheses have been omitted. To be sure, this equation is ambiguous without them. Figure 4-2 illustrates three possible interpretations of this equation; in each one the variables are grouped differently.

The equation, restated in Fig. 4-2A, could be interpreted according to the priority rule: variables B and C are ANDed first; this is the *inner term*, term 1. The result, which is the entire expression

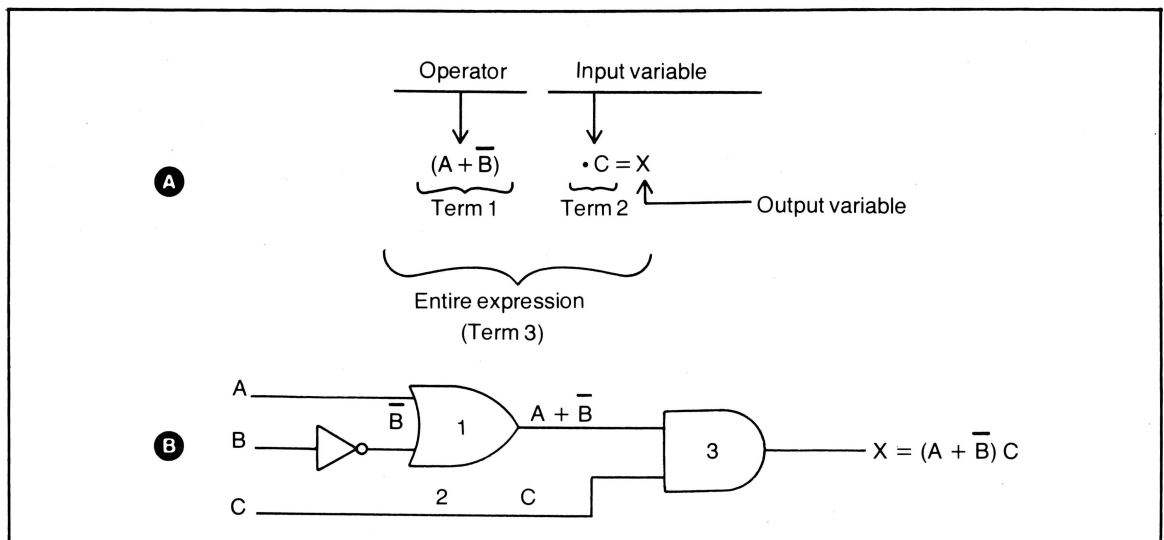


Fig. 4-1. Correspondence between elements in an equation, A, and its schematic representation, B.

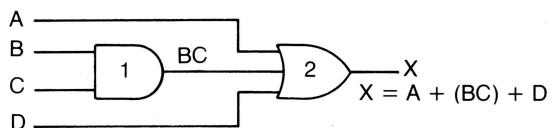
A

$$A + B \cdot C + D = ?$$

B

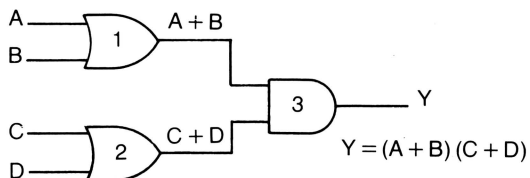
$$A + (BC) + D = X$$

Diagram showing the equation $A + (BC) + D = X$ with a bracket labeled '1' over (BC) and a bracket labeled '2' under $A + D$.

**C**

$$(A + B) (C + D) = Y$$

Diagram showing the equation $(A + B) (C + D) = Y$ with a bracket labeled '1' over $(A + B)$, a bracket labeled '2' over $(C + D)$, and a bracket labeled '3' under the entire expression.

**D**

$$A + B(C + D) = Z$$

Diagram showing the equation $A + B(C + D) = Z$ with a bracket labeled '2' over $(C + D)$, a bracket labeled '1' over $B(C + D)$, and a bracket labeled '3' under $A + B(C + D)$.

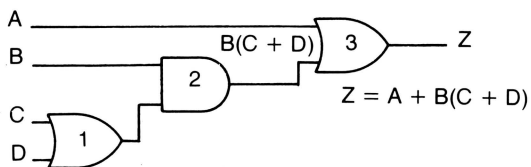


Fig. 4-2. Three different interpretations of a Boolean equation. See text.

or term 2, is ORed with A and D to form the final output X. See Fig. 4-2B. You can see the correspondence between the labeled terms in the equation and the gates in the schematic to the right. (Incidentally, the entire expression is an OR-form expression, of which I will say more later.)

Two other interpretations are shown in Fig. 4-2C and D. The schematic circuit diagrams for these equations, drawn to the right, are quite different from the schematic for Fig. 4-2B. This is a consequence of the grouping into different terms, each of which is again numbered in both the equation and in the schematic for comparison.

More importantly, there is really no way of expressing these latter two relationships other than by grouping by parentheses. This is why the priority rule is not all that useful in the practical writing and interpretation of Boolean equations.

By the way, there is a fourth grouping not listed in Fig. 4-2. See if you can figure it out.

Another property of Boolean equations is their *form*. Two forms exist: the *AND-form* and the *OR-form*.

In the AND-form, two terms (or two variables) are ANDed together to form the whole expression. Equations 4-3a through 4-3c are all AND-form expressions.

$$AB = W \quad \mathbf{4-3a}$$

$$(A+B)(C+D) = X \quad \mathbf{4-3b}$$

$$(A+BC)(\overline{CD}+(A(B+C))) = Y \quad \mathbf{4-3c}$$

The simplest example is Equation 4-3a, which is the two-variable AND relationship you studied in the last chapter. Equation 4-3b is another AND-form equation, in fact the same one is given in Fig. 4-2C. This is an AND-form because the two main terms, (A+B) and (C+D), are ANDed together to yield the final expression. Similarly, the two main terms of Equation 4-3c are ANDed together, making this another AND-form relationship. This last equation has lesser, or inner, terms of course: (A+BC) is an OR-form, BC and (A(B+C)) are both AND-forms, and so on. But again, it is the two largest terms that determine the form of an equation, not the inner terms.

Another way of understanding this is to look back at Fig. 4-2 and note that the last gate in the schematic diagram indicates the form of the equation it represents. For instance, Fig. 4-2C illustrates an AND-form relationship, as can be seen from gate 3 in the schematic. This graphic line of reasoning would apply to most any combinational equation.

For reasons similar to the previous equations, the following set of equations are of the OR-form:

$$A+B = L \quad \mathbf{4-4a}$$

$$(\overline{AB}) + (CD) = M \quad \mathbf{4-4b}$$

$$(\overline{A(B+D)}) + (C+D+E) = N \quad \mathbf{4-4c}$$

Again, the two largest terms in each expression to the left of the equals sign determines the form of the equation.

Finally, there are alternate names each for the AND-form and OR-form type of expressions. The AND-form is also known as the *product of sums form* (POS), and the OR-form is also referred to as the *sum of products form* (SOP).

These names have their origin in the analogy between the dot (•) for AND and the dot for multiply, and between the plus (+) for OR and the plus for addition. These names are therefore slightly inaccurate but understandable, nonetheless. Expressions which are of the AND-form are also called *minterms*, (A(B+C)) for instance. OR-form expressions may be called *maxterms*; e.g., (A+BC) is a maxterm.

The whole point of Boolean algebra is to relate the inputs and outputs of a digital circuit in a condensed, easy-to-understand form. Boolean variables are related by Boolean operators. These variables and operators are grouped into terms, which are then related to one another in turn by other operators. The form of the resultant expressions can be characterized as a SOP/maxterm form or as a POS/minterm form. The value of the expression, regardless of its form, takes on a single logic or Boolean value, 1/0, true/false, or high/low. This is equivalent to the output of a combinational logic circuit, and can be represented by a single variable.

So far you have only a collection of terms and

Table 4-1. Summary of Boolean Rules.

Postulates

1	$\overline{\overline{A}} = A$	Double Negative
2a	$A \cdot 0 = 0$	
b	$A + 1 = 1$	Nullity (disabled)
3a	$A \cdot 1 = A$	
b	$A + 0 = A$	Identify (enabled)
4a	$A \cdot \overline{A} = 0$	
b	$A + \overline{A} = 1$	Complements
5a	$A \cdot A = A$	
b	$A + A = A$	Idempotency (IS)

Laws

6a	$A \cdot B \cdot C = B \cdot A \cdot C$	
b	$A + B + C = B + A + C$	Commutative Law
7a	$A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C$	
b	$A + (B + C) = (A + B) + C = A + B + C$	Associative Law
8a	$A \cdot (B + C) = AB + AC$	
b	$A + B \cdot C = (A + B) \cdot (A + C)$	Distributive Law
9a	$A + A \cdot B = A$	
b	$A \cdot (A + B) = A$	
c	$A \cdot (\overline{A} + B) = AB$	
d	$AB + \overline{B} = A + \overline{B}$	
e	$\overline{A}B + B = A + B$	
f	$\overline{A}B + A = A + B$	Absorption Rules

De Morgan's Theorem

10a	$\overline{A \cdot B} = \overline{A} + \overline{B}$	POS to SOP
b	$\overline{A + B} = \overline{A} \cdot \overline{B}$	SOP to POS

Note: Priority Convention—AND before OR. See text.

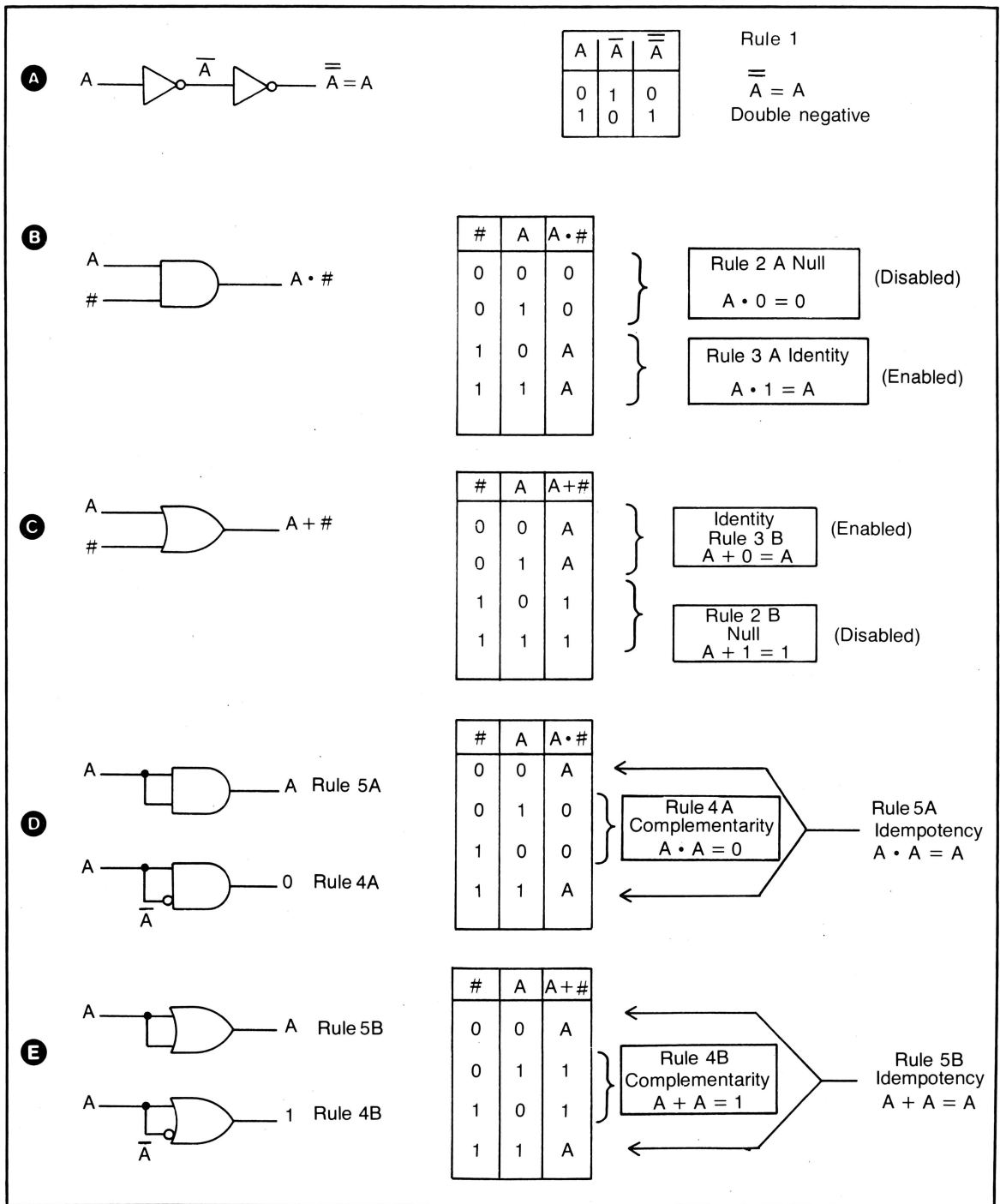


Fig. 4-3. Five Boolean postulates, with their truth table and schematic representation. A) Double negative. B and C), Null and Identity rules follow from basic AND and OR properties, and formalize the enable/disable idea. C and D), The rules of Complements and Idempotency, which also follow basic properties, are illustrated.

definitions. However, the nomenclature is necessary when I discuss Boolean rules of simplification and when I explain how to translate between the various forms of logical representation: truth table, logic equation and schematic diagram.

Boolean Postulates

A Boolean equation is a shorthand notation for expressing the logical relationships among signals in a digital circuit. Because the equations are algebraic, they can be manipulated and simplified according to a set of rules. The primary benefit of this is that the resulting circuits are also simplified with considerable savings in chip count, cost, and construction and troubleshooting time.

Table 4-1 summarizes the rules of Boolean algebra because they are analogous to the laws of this and the next few sections. The first five rules might be termed Boolean *postulates* as they derive from the basic definitions of AND, OR and NOT. Whatever purists may call them (axioms, definitions, or properties), these five rules are the essential givens of the algebra.

Rules 6, 7, and 8 are called the laws of Boolean algebra because they are analogous to the laws of the same name in conventional algebra.

Rule 9 is a set of relationships under the heading Absorption. These are not fundamental relationships in the sense of the preceding rules. In fact, their proof depends upon the application of the prior rules. But they are occasionally useful in equation simplification and are included in the table for this reason.

Rule 10 is De Morgan's Theorem, which relates AND-form and OR-form types of expression. The three laws and this theorem get separate attention later on.

Turning to the first five postulates or properties, you will recognize the first immediately. This Rule 1 is the double negative or NOT-NOT property mentioned in the last chapter. This rule and its associated truth table and schematic, are illustrated in Fig. 4-3A. The truth table is actually the *proof* of this rule, as you can see by inspection. Proving a logical relationship by truth tables is known as proof by perfect induction. This postulate is perhaps the

simplest example.

Rule 2, Nullity, and Rule 3, Identity, are illustrated in Fig. 4-3B and C. Both rules are notable in that they contain only one variable, just like Rule 1; the other factor in the expression is a binary number, 0 or 1. The *Null* postulate is so named because the sole variable has no effect on the output or result. This is seen in the truth tables for the AND and OR functions in Fig. 4-3B and C. Conversely, the *Identity* postulate, Rule 3, states the conditions when the sole variable does determine output. If all of this sounds familiar, it should be, because these two rules are simply formal restatements of the enable/disable states of AND and OR devices covered earlier. For instance, Rules 2B and 3B are expressions of the enabled and disabled conditions for the OR device. Again, you can prove the validity of these two rules by inspecting the truth tables for AND and OR in the figure.

Rule 4, Complementarity, and Rule 5, Idempotency, have two variable terms in their expression: A with itself or its inverse. The *Complementarity* rule (or complements) ANDs or ORs the variable A with its inverse, as shown in Fig. 4-3D and E. The result is either 0 or 1, respectively, as you would expect from the definitions of AND and OR. In the expression for Rule 5, the *Idempotency* postulate, the variable A is ANDed or ORed with itself, the result being A. Rule 5 therefore is the formal statement of the Boolean IS function, one that is suggested by the schematics shown in the figure. (Remember the noninverting buffer.)

Figure 4-3 is certainly not an illustration to glance at casually. Study it again if necessary. Understand why these five postulates are true on the basis of the truth tables (proof by perfect induction). Relate the enable/disable and noninverting buffer functions to the appropriate rule. It is not necessary that you carry all of this in your head all the time. But it is necessary that you understand it thoroughly once. This should not be too difficult because these postulates have, for the most part, already been demonstrated on the breadboard.

Boolean Laws and Absorption

The three Boolean laws, Rules 6, 7, and 8 in

Table 4-1, are similar to the laws of the same name in ordinary algebra. They are better understood when related to actual circuit diagrams, as done in Fig. 4-4.

In Fig. 4-4A, the two forms of the Commutative Law, Rule 6, state that you can rearrange the variables (or terms) around an AND-form or OR-form expression without changing the value of the expression. Relabeling the inputs to an AND device is not going to change the output: all inputs must be high/1 for the output to be high. Likewise, shifting the input variables around to an OR device does not change the basic function: the output will be high if any one of the inputs assumes a value of binary 1.

Rule 7, the Associative Law, states that you can group the variables or terms in a pure minterm (AND-form) or pure maxterm (OR-form) expression without changing the result of the expression. The sense of this is appreciated by examining Fig. 4-4B. You can, for instance, use two 2-input AND devices to do the job of a single 3-input AND device. Likewise for the OR device shown. This often has a direct practical application in actual circuits in that it may reduce package count. (Caution: This does not work for NAND and NOR gates.)

Rule 8, the Distributive Law, states how a variable is distributed over an AND or OR operator. Figure 4-4C gives the schematic form of the law. From a practical standpoint, the law could be directly applied to the circuit shown, thereby reducing device count: that is, variable A is factored out, eliminating one gate from the circuit implementation. From an algebraic standpoint, it shows how to expand a minterm/POS expression of the general form $A(B+C)$ and how to expand a maxterm/SOP expression of the form $A + BC$. The expanded form often allows for regrouping of terms and simplification of more complex equations. Rule 8 is, in fact, particularly useful in simplifying Boolean equations; this is illustrated below and in later sections.

The set of relationships under the heading of Rule 9, the rules of Absorption, are derivative rather than fundamental Boolean properties. They are sometimes useful in simplifying equations. Rather than accept them on faith, you may want to

prove their validity. To do this, you will use a few of the eight prior rules just stated. For example, how would you prove Rule 9a— $A+AB=A$? Try it before looking at the solution.

$$\begin{aligned} A + AB &= A(1+B) \text{ Distributive/Rule 8} \\ &= A(1) \text{ Null/Rule 2} \\ &= A \text{ Identity/Rule 3} \end{aligned}$$

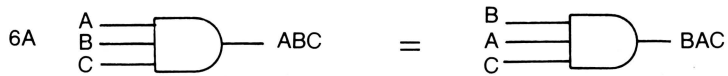
A more difficult example is the proof of Rule 9e— $A\bar{B}+B=A+B$.

$$\begin{aligned} A\bar{B} + B &= A\bar{B} + B(A+1) \text{ Null/Rule 2 and} \\ &\quad \text{Identity/Rule 3 (A trick} \\ &\quad \text{by which B is ANDed} \\ &\quad \text{with 1, leaving its value} \\ &\quad \text{unchanged)} \\ &= A\bar{B} + AB + B \text{ Distributive/Rule 8} \\ &= A(\bar{B} + B) + B \text{ Distributive/Rule 8} \\ &\quad \text{again} \\ &= A(1) + B \text{ Complements/Rule 4} \\ &= A + B \text{ Identity/Rule 3} \end{aligned}$$

Another general method of proof already introduced is the proof by perfect induction, which involves the use of truth tables. Taking Rule 9e again, we can demonstrate its validity by means of a four-line truth table, shown in Table 4-2. Constructing this or any other truth table for combinational equations involves the following steps:

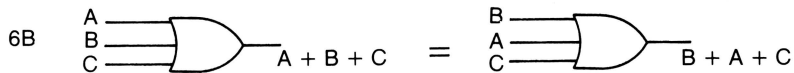
1. Begin by writing all the variables, inverted and noninverted, in the expression in the left-most columns.
2. Proceed by writing in the smaller terms, AND-form and OR-form.
3. Continue building the table from left to right, using larger terms until you have the entire expression in the right-most column.
4. If you are doing a proof by perfect induction, an additional column to the extreme right should be included. This last column specifies the right-hand side of the equation to which the equality is being tested.

As you can see in Table 4-2, columns 1 and 4

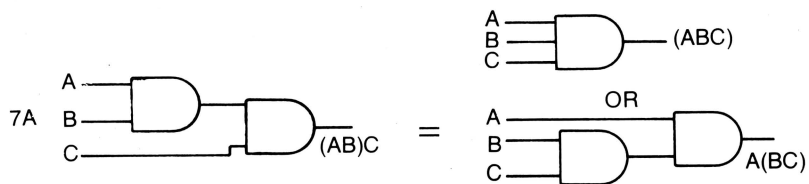


A

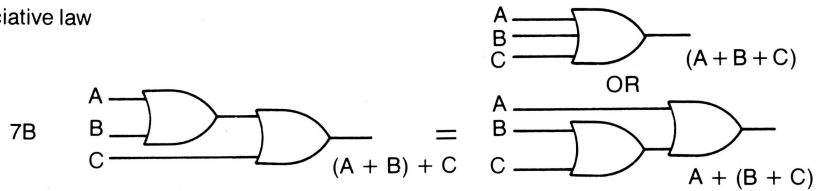
Commutative law



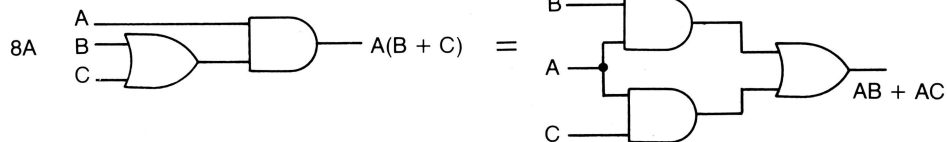
B



Associative law



C



Distributive law

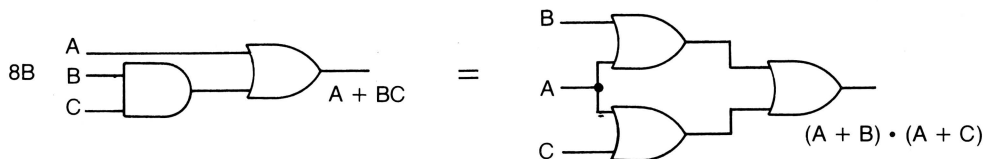
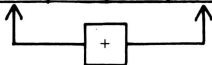


Fig. 4-4. The Laws of Boolean algebra, with illustrative circuit symbology. They are analogous to ordinary algebraic laws.

Table 4-2. Proof by Perfect Induction for Rule 9e.

B	A	\overline{B}	$A\overline{B}$	$A\overline{B}+B=A+B$	
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

← Rule 9e



are used to create the left-hand side of the equation, columns 1 and 2 the right-hand side: the equality between $A\overline{B}+B$ and $A+B$ are proven in the final two columns.

This method of building truth tables is very useful, because it is usually the first step in designing digital circuits from a list of input and output specifications. Sometimes, it is useful to follow this procedure to double-check (prove) your results after having simplified a complex equation.

De Morgan's Theorem

Rule 10 in Table 4-1 is De Morgan's Theorem. This important rule shows how to change a minterm/AND-form expression into a maxterm/OR-form expression, and vice versa. Rule 10a states that the NAND expression (POS/minterm) is equivalent to an OR-form expression. It says that an NAND device is equivalent to an OR device with inverted inputs. See Fig. 4-5A.

Rule 10b states that the NOR function, expressed as a maxterm/SOP form, can also be expressed as a minterm/POS form. NOR is equivalent to an AND device with negated inputs, as in Fig. 4-5B.

The gate equivalence just described can be proved most conveniently by proof by perfect induction. The truth tables of Fig. 4-5A and B show the validity of the De Morgan Theorem for the simplest case of NAND and NOR. Naturally, this relationship holds for more complex expressions.

Generally, the application of the Theorem can be described as a three-step process:

1. Invert each of the major terms in the expression.
2. Change the operators between each of these terms from AND [\bullet] to OR [$+$] or vice versa, as appropriate.
3. Invert the entire resulting expression.

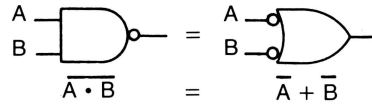
If these steps are applied to an AND device, Fig. 4-5C, you can see that it is equivalent to an OR device whose inputs and outputs are inverted. Similarly, an OR function is performed by an AND device with inverted inputs and outputs. The Theorem, then, is simply a formal statement of the gate equivalence concept that was mentioned briefly in the last chapter. It is only a matter of extending this concept a bit further to see how *gate transformation* can be accomplished. That is, using only a single NAND or NOR package you can implement all the basic combinational logic functions.

For example, how would you use De Morgan's Theorem to transform a NAND package into a circuit performing the NOR function? Remember, you can use only the devices on a single NAND package.

Figure 4-6 shows how this is done. In step 1, the NAND/negated-input OR relationship is restated. In step 2, the inputs are again inverted. This, by the double inversion postulate (Rule 1), creates an OR device, indicated in step 3. In step 4, the output is inverted, creating the desired NOR function. The equations are to the left, and the corresponding changes in OR symbology are in the middle. The column on the right shows the actual physical changes that are made to a 2-input NAND device. As you can see, the final NOR function requires four NAND devices, an entire NAND 74LS00 package. In practical situations, you would rarely implement such a transformation unless you were breadboarding and had no LS02 NORs available. You might have occasion to use 3/4 of an LS00 to create the OR function in step 2 more frequently, however. Remember that you could also use LS04 inverters in combination with a single NAND to obtain either the OR or NOR functions.

A

B	A	\bar{B}	\bar{A}	$\overline{A \cdot B}$	$\bar{A} + \bar{B}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

**B**

B	A	\bar{B}	\bar{A}	$\overline{A + B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

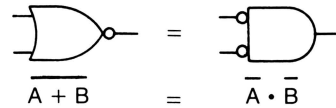
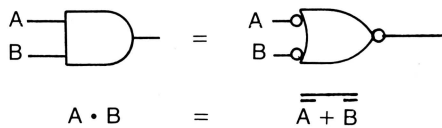
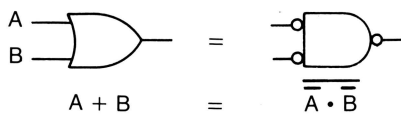
**C****D**

Fig. 4-5. De Morgan's Theorem shows the equivalence between minterm/AND-form expressions and maxterm/OR-form expressions, and their corresponding gate symbols. A) NAND is equivalent to negated input OR. B) NOR to negated input AND. C) AND to an OR with negated inputs and output. D) OR to an AND with negated inputs and output.

Finally, gate transformation is summarized in Fig. 4-7. In 4-7A, the use of NAND to create AND, OR and NOR is indicated. The NOR device is just as flexible, and can be used to obtain OR, AND and NAND functions, as in Fig. 4-7B. Both NAND and NOR can be transformed into NOT and IS buffers as done earlier. Theoretically then, either device could be used exclusively to build moderately complex circuits!

De Morgan's Theorem does more than give us

a means of transforming gates, of course. The application of the Theorem, along with the other rules, to equation simplification and circuit design will be explored in the next and following sections.

Examples of Simplification

Included in this section are two short examples of combinational logic equations that may be simplified using the Boolean rules. Here are two

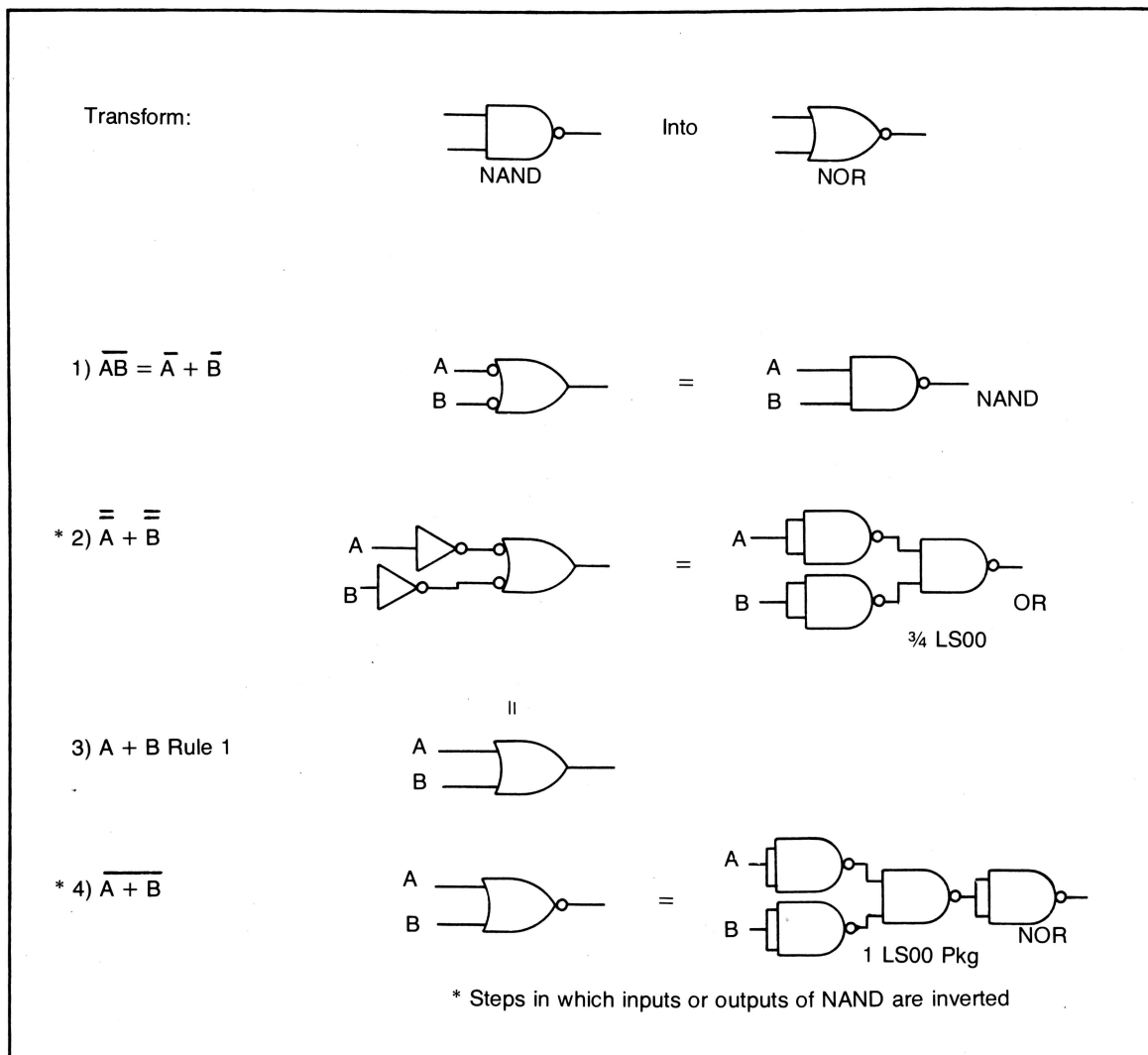


Fig. 4-6. De Morgan's Theorem can be used to transform one device function into another. Here, a NAND package is configured to provide OR and NOR functions, using three and four NAND devices, respectively.

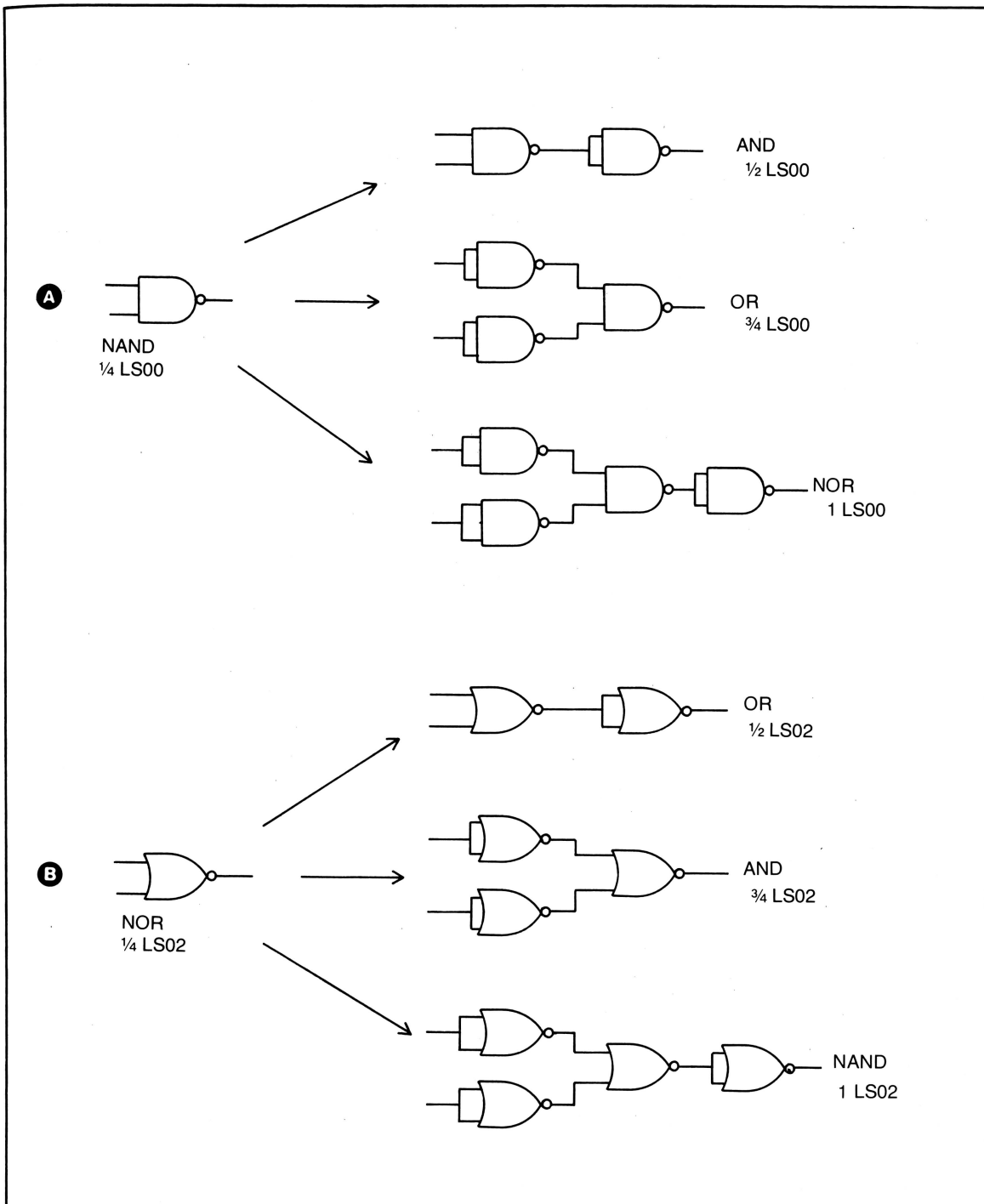


Fig. 4-7. Gate transformation is one important consequence of De Morgan's Theorem, and is summarized here for A) NAND and B) NOR devices.

equations that you should try working out before looking at the solutions.

$$\overline{\overline{A(B+C)} + \overline{A}} \\ ((A(\overline{A}+B))B + \overline{C})\overline{B}$$

The solution for the first equation is as follows:

$$\begin{aligned} \overline{\overline{A(B+C)} + A} &= A + \overline{\overline{B+C}} + \overline{A} && \text{De Morgan/Rule 10} \\ &= (A + \overline{A}) + \overline{B+C} && \text{Associative/Rule 7} \\ &= 1 + \overline{B+C} && \text{Complements/Rule 4} \\ &= 1 && \text{Null/Rule 2} \end{aligned}$$

Circuit design involves writing and simplifying equations. However, it would be rather embarrassing to implement this particular function in a circuit in its nonsimplified form, only to discover that all you needed was a +5 volt logic high. If this is, in fact, the function you require in a circuit, then simply tie the line calling for this function to Vcc (+5 V) through a 1 K resistor. The second example is solved in the following manner:

$$\begin{aligned} ((A(\overline{A}+B))B + \overline{C})\overline{B} &= ((\overline{A}A + AB)B + \overline{C})\overline{B} && \text{Distributive/Rule 8} \\ &= ((0 + AB)B + \overline{C})\overline{B} && \text{Complements/Rule 4} \\ &= (ABB + \overline{C})\overline{B} && \text{Identity/Rule 3 and} \\ &= ABB\overline{B} + \overline{C}\overline{B} && \text{Distributive/Rule 8} \\ &= \overline{C}\overline{B} && \text{Distributive/Rule 8} \\ &= \overline{B+C} && \text{Complements/Rule 4 and} \\ & && \text{Identity/Rule 3} \\ & && \text{De Morgan/Rule 10} \end{aligned}$$

The Distributive Law will be used repeatedly when there are multiple parentheses giving *nested* terms, as in this example.

RELATING INPUTS AND OUTPUTS

We can use a truth table to generate an equation. From an equation we can generate a schematic. The opposite sequence is also true. From an equation we can generate a truth table.

Equations from Truth Tables

How do you convert a truth table into a logic equation? The method is straightforward:

1. From each line on the truth table write a minterm. This consists of the ANDed input variables on that line. Each minterm may be equal to 0 or 1, depending on the value in the output column.

2. The equation is written by taking the sum of those minterms. That is, by ORing them together and setting the resultant SOP (sum of products) expression equal to the output, you have the equation.

To illustrate the concept of equation building from truth tables, let's take the AND and OR functions with which you are already familiar. In the case of AND, each line on the truth table in Fig. 4-3A corresponds to a minterm. The value of that minterm, its output, is either 0 or 1 (true or false). For the 2-input AND function there are four minterms labeled m0, m1, m2, and m3. Line 0 would read: "Minterm 0, in which A and B are both zero, is false (low/binary 0)."

There are three minterms which have a zero value in this truth table: m0, m1, and m2. This is the same as saying that the output (X) of the AND device has a value of zero for any one of these three states. In equation form this becomes:

$$\overline{A}\overline{B} + A\overline{B} + \overline{A}B = 0 = X \quad 4-5a$$

Alternatively, you could write the equation for the function expressed in the truth table by using the minterm(s) which have a value of true/high/1. (This is the same as asking, "For what set(s) of input values is the output of the device high/binary 1?"). There is only one such minterm, m3. The equation is the AND relationship:

$$AB = 1 = X \quad 4-5b$$

Equation 4-5b states that X is true/high/1 only when both inputs are high.

When writing an equation from a truth table, then, you must use the minterms having output states of either 1 or 0. Choose the states in the minority so that there are fewer minterms in the equation to deal with.

In the case of OR, Table 4-3B, the situation is

Table 4-3. SOP HIGH and SOP LOW Equations for A) AND, B) OR.

A

Minterm#	B	A	X = AB	Minterm Exp
m0	0	0	0	$\bar{A}\bar{B} = 0$
m1	0	1	0	$A\bar{B} = 0$
m2	1	0	0	$\bar{A}B = 0$
m3	1	1	1	$AB = 1$

False Minterms
Low Output

True Minterm, High Output

OR ↗ Low Out:
↘ High Out:

$$m0 + m1 + m2 = 0 \rightarrow \bar{A}\bar{B} + A\bar{B} + \bar{A}B = 0$$

$$m3 = 1 \rightarrow AB = 1$$

$$AB = 1$$

B

Minterm #	B	A	X = A + B	Minterm Exp
m0	0	0	0	$\bar{A}\bar{B} = 0$
m1	0	1	1	$A\bar{B} = 1$
m2	1	0	1	$\bar{A}B = 1$
m3	1	1	1	$AB = 1$

False Minterm, Low Output

True Minterms, High Output

OR ↗ Low Out:
↘ High Out:

$$m0 = 0 \rightarrow \bar{A}\bar{B} = 0 \text{ or } A + B = 1$$

$$m1 + m2 + m3 = 1 \rightarrow A\bar{B} + \bar{A}B + AB = 1$$

reversed. It is the high-valued minterms which are in the minority; only one, m0, is low. Equation 4-6a defines all the low valued states of the output in the truth table:

$$\begin{aligned}\overline{A \cdot B} &= 0 && \text{(from the truth table, m0)} && \mathbf{4-6a} \\ \overline{A + B} &= 0 && \text{De Morgan/Rule 10} \\ A + B &= 1 && \text{Negate both sides (note that} \\ &&& 0=\overline{1} \text{ and } 1=\overline{0})\end{aligned}$$

You could also combine all of the high-valued minterms into Equation 4-6b, which is an SOP expression:

$$\begin{aligned}\overline{A} \overline{B} + \overline{A} B + A B &= 1 \text{ (sum of minterms m1, m2, m3)} && \mathbf{4-6b} \\ A(B + \overline{B}) + \overline{A} B &= 1 \text{ Commutative/Rule 6 and} \\ &&& \text{Distributive/Rule 8} \\ A + \overline{A} B &= 1 \text{ Complements/Rule 4 and} \\ &&& \text{Identity/Rule 3} \\ A + B &= 1 \text{ Absorption/Rule 9f (yes, it is} \\ &&& \text{actually used on occasion).}\end{aligned}$$

This is a longer route to the same result!

What about the case in which there are an equal number of high and low output states? In such instances both the SOP high equation (sum of minterms with values of 1) and SOP low equation will have the same number of minterms. As a convenience, you would normally choose the SOP high equation as this usually eliminates one step in simplification. As an example, take the function described by the truth table in Fig. 4-8A. From this table two equations can be written.

$$\overline{A} \overline{B} \overline{C} + \overline{A} \overline{B} C + \overline{A} B \overline{C} + A B C = 1 = W \quad \mathbf{4-7a}$$

$$\overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + A B \overline{C} + \overline{A} B C = 0 = \overline{W} \quad \mathbf{4-7b}$$

If you tried to implement Equation 4-7a directly, term for term, you would require five inverters, four 3-input AND gates, and one 4-input OR. This comes to a total of ten IC devices. If you

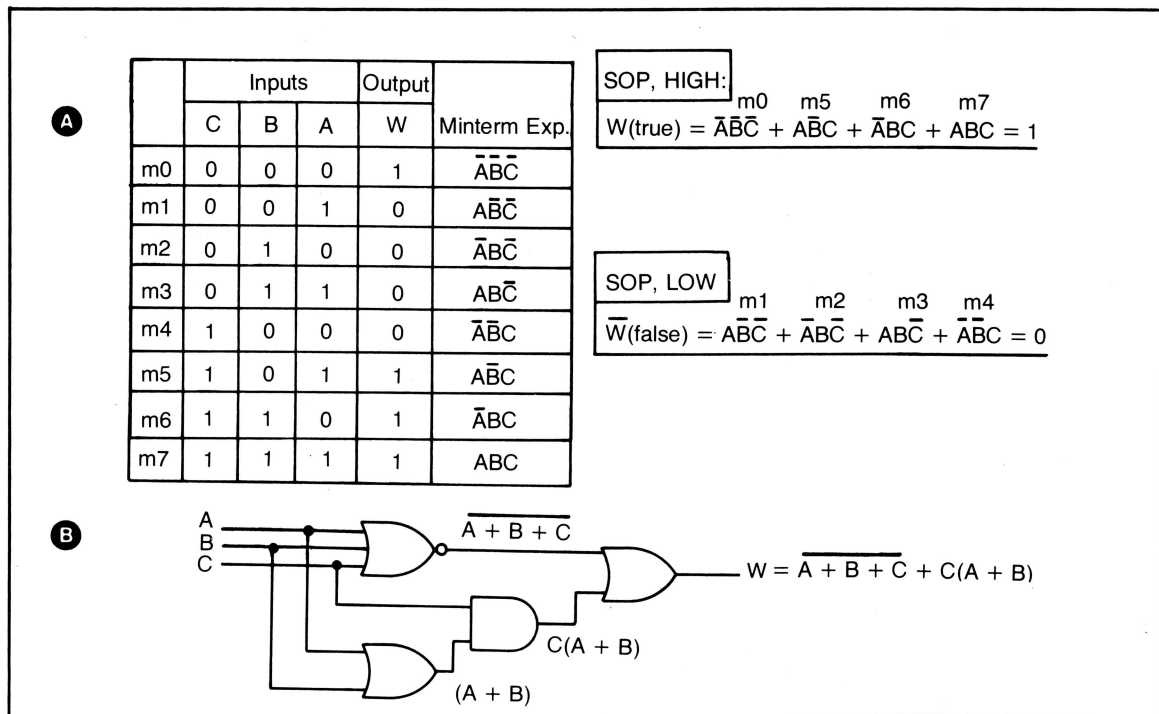


Fig. 4-8. In some Boolean relationships the truth table has an equal number of high and low outputs, as in (A). Write the sum of products (ORed minterms) expression for the output high state, i.e., the SOP high equation, output true. (B) shows the simplified expression and associated schematic for the function in (A).

simplified the equation first (try it) you would reduce gate count dramatically. Equation 4-8 is the simplest solution, requiring the fewest gates for the function expressed in the truth table. Only four gates are required.

$$\overline{A+B+C} + C(A+B) = 1 = W \quad 4-8$$

The logic circuit for this expression is given in Fig. 4-8B.

Finally, there is the more general class of functions with multiple outputs.

The truth tables expressing these functions likewise will have more than one output column.

The usual approach in translating such truth tables into logic equations is to write a separate equation for each output column. Each equation will have one or more minterms which are ORed together and equated to one of the output columns. Another equation is written for the next output column, and so on. (SOP high or SOP low type equations, or a mixture of both, are used.) The main problem is how to use the inputs in the most efficient fashion, without having to construct a separate circuit for each equation. Multiple output circuits are fairly common, so the problem is a very practical one. To illustrate these points, you will examine a 3-input *majority detector*, shown in Fig. 4-9.

Referring to the truth table of Fig. 4-9A, you'll note that one of the output columns, MAJ, is high/1 when two or more of the inputs are high. The relationship expressed is that of a majority detector, and the equation is written as a SOP high expression (m3, m5, m6, m7) equated to the output, MAJ, as indicated in Fig. 4-9B.

If you require another output separate line to indicate unanimity (UN), use the minterm(s) that are high when all the inputs are high. Since there is obviously only one such minterm, m7, it is easy to write the equation (Fig. 4-9B).

Perhaps a third output line signaling minority is needed as well. Before writing the 4-minterm expression, observe from the truth table that the minority state, MIN, is simply the inverse of MAJ: MIN=MAJ.

Now, using the equations in Fig. 4-9B, draw the circuit diagram. First draw the schematic for the nonsimplified MAJ equation. To have unanimity output, take the output of the gate labeled m7 directly. For MIN, just invert the output of the 4-input OR gate. See Fig. 4-9C. Circuit overhead for these two extra outputs is negligible.

Not all multiple output digital circuits are so straightforward. Combinational circuits which generate binary codes from single-bit lines—keyboard encoders that generate ASCII code, for instance—are much more complex than the example above. So are other functions, such as decoders, multiplexers, arithmetic processor chips, etc. These functions fall in the range of MSI complexity. However, you will occasionally need to design a small circuit with perhaps two or three outputs. Often, SSI packages will serve the need quite well. So, it is good if you see that these little custom circuits are well within your design ability, even at this stage.

Drawing Schematics from Equations

If you have followed the examples so far, the process of converting a simple combinational equation into a schematic diagram probably seems quite straightforward. A few guidelines will sharpen this ability even further.

When drawing a schematic diagram from a logic equation, you will use the left-hand expression in the equation and proceed as follows:

1. From the inner to the outer term in the equation, and
2. From the left to right and from top to bottom in the schematic.

Two equations will illustrate the methods. Draw your own logic circuit before reading the solutions.

$$(A+\bar{B})C + BD = X \quad 4-9$$

$$AB + \bar{C}(D+EF) = Y \quad 4-10$$

In Equation 4-9 (see Fig. 4-10A) there are three *levels* or layers of symbols in the circuit diagram. The left-most symbols correspond to the

	Inputs			Outputs		
	C	B	A	MAJ	UN	MIN
m0	0	0	0	0	0	1
m1	0	0	1	0	0	1
m2	0	1	0	0	0	1
m3	0	1	1	1	0	0
m4	1	0	0	0	0	1
m5	1	0	1	1	0	0
m6	1	1	0	1	0	0
m7	1	1	1	1	1	0

Note: $\overline{\text{MAJ}} = \text{MIN}$

$\text{MAJ} = m3 + m5 + m6 + m7 = \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + ABC = 1$
 $\text{MIN} = m0 + m1 + m2 + m4 = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC = 1$
 $\text{UN} = m7 = ABC = 1$

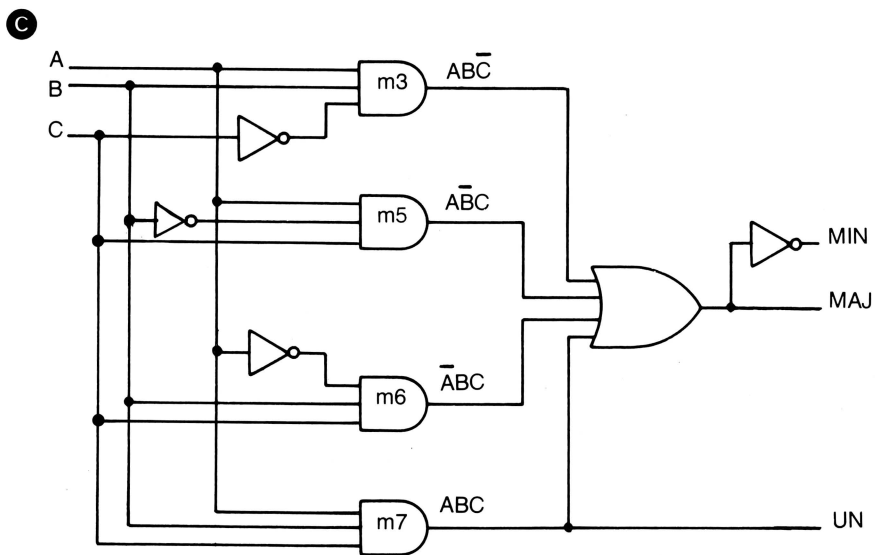


Fig. 4-9. A) Truth table for a multiple output function, the majority detector. B) Equations for each of the three outputs. C) Unsimplified implementation of these equations. See text for discussion.

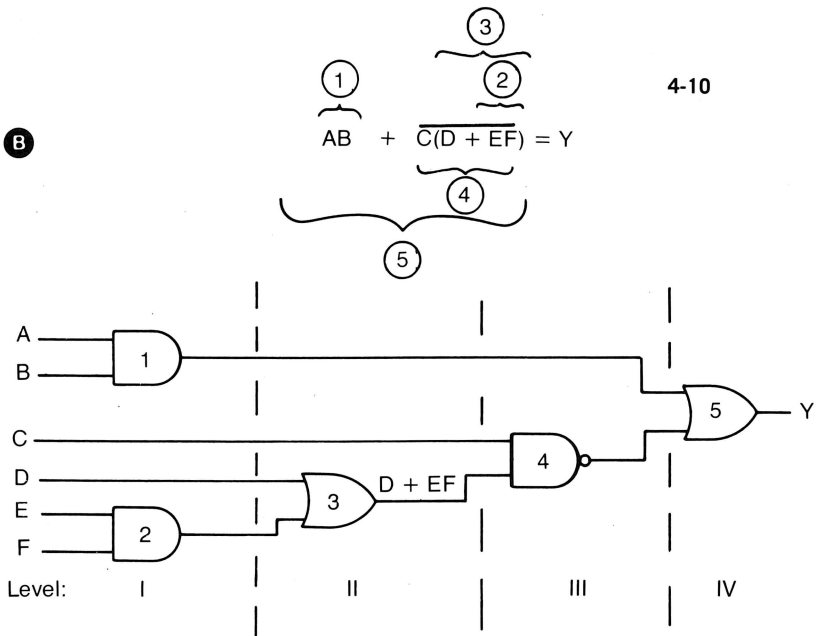
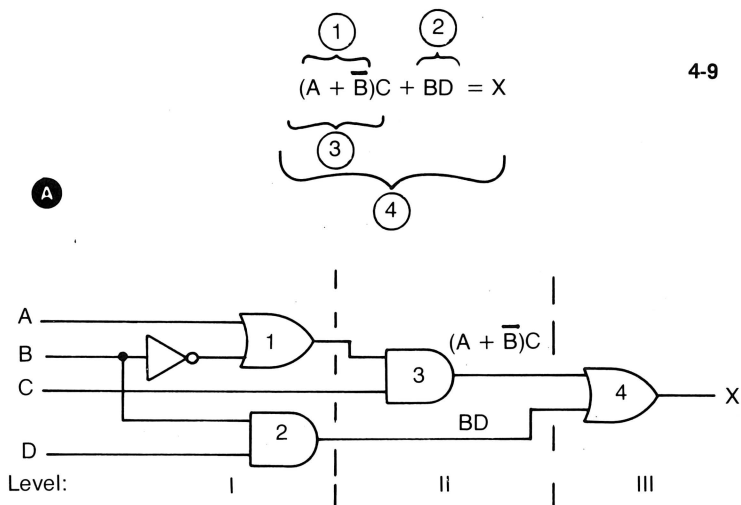


Fig. 4-10. Building schematic circuit diagrams from logic equations. See text for discussion of A and B.

inner-most terms in the equation. These inner terms may be minterm (AND-form) or maxterm (OR-form). In fact, this is the case here: term 1 is a maxterm, and term 2 is a minterm; both are inner-most in the expression and are left-most in the schematic, at level I.

As you construct the diagram, going from left to right, you will use larger and larger terms. At level II there is only one term, term 3, and at level III there is again only one term, term 4. This step-wise, inner-to-outer, left-to-right progression is illustrated by the numbered brackets over the equation terms, and by the numbered devices in the schematic. Further, the level in the circuit, which reflects the progression from smaller to larger terms, is indicated by Roman numerals in the schematic diagram.

Let me add a few other points about writing schematics from equations. First, the input lines are labeled alphabetically from top to bottom. While this may result in a few more overlapped lines than would be the case in different arrangements, this alphabetic ordering is more logical and easier to follow. (Naturally, when using meaningful signal names such as ENB, DAT, CLOCK, etc., you would apply them to the respective lines as appropriate.)

Second, you do not normally count inversion of a variable or term as an additional level, because inversion does not combine variables or terms into larger terms. For instance, in this example, the use of variable B in both inverted and noninverted form does not add an extra level to the circuit.

Converting Equation 4-10 into a circuit diagram follows the same rules just mentioned. As illustrated in Fig. 4-10B, there are four levels in this combinational relationship. The two inner-most minterms, 1 and 2, are at level I. Term 2 is ORed with variable D at level II. Note that variable C does not enter into this Boolean expression until level III. Terms 1 and 4 are finally ORed at the last level, making the entire expression a sum of products or OR-form type. This verbal description is obviously more obscure than the diagrammatic one, where you can again readily see the step-wise buildup of the equation as you proceed from left to right.

In short, identify inner and outer terms in the logic equation first. Begin the circuit diagram by labeling the input lines alphabetically. Starting with the inner-most terms, build the expression in schematic form using appropriate logic symbols. As you draw the circuit diagram, you may find it helpful to identify the gate and term numbers, and to write in the terms on the device lines, as done in Fig. 4-10.

NAND/NAND Logic

From De Morgan's Theorem, you learned about the flexibility of NAND and NOR packages. Either could be configured to perform all the basic combinational logic functions. Theoretically, entire digital systems could be constructed from nothing but SSI NAND (or NOR) devices. Real-world considerations such as power consumption, speed, circuit noise, timing problems, and space requirements would make such a system impractical. However, simple combinational circuits can be built entirely of SSI components. This is sometimes desirable for implementing custom functions for which off-the-shelf MSI chips are not readily available. In such cases, you can use NAND or NOR devices exclusively, with considerable savings in package count and cost.

NAND/NAND logic—circuits using only NAND devices—is usually preferable over NOR/NOR logic if the number of devices in the two implementations is the same. This is because the electronics of the NAND circuit is a bit simpler than that for the NOR (one less transistor, usually). As a result, the speed of the device is slightly higher. Comparable subfamilies of NAND are also a little cheaper. These differences are minor, but sufficient to give NAND the edge.

An example of the utility of NAND/NAND logic for reducing package count is presented in Fig. 4-11. The combinational function performed by the circuit is the same as that expressed by Equation 4-9. The circuit shown in Fig. 4-11A is in fact the same as that in Fig. 4-10A. It employs five gates, and this involves three IC packages, as indicated to the right of the schematic. Each package requires its own board space, power and ground

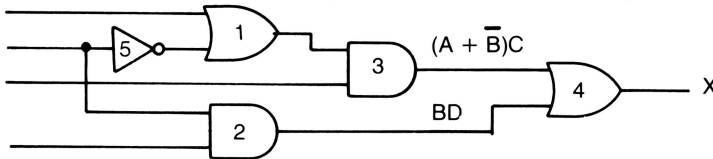
$$(A + \bar{B})C + BD = X$$

4-9

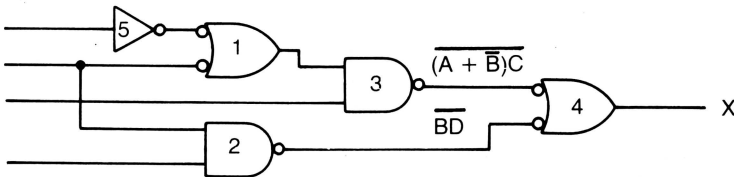
Gate #	IC Package
1, 4	1/2 LS32
2, 3	1/2 LS08
5	1/6 LS04

Total package count = 3

A



B



Gate #	IC Package
1-4	1-LS00
5	1/6 LS04 or 1/4 LS00

Total package count = 2

Saving 33 1/3%

Fig. 4-11. Using NAND/NAND Logic to minimize IC package count. A) Literal mixed gate implementation of Equation 4-8. B) Using NAND/NAND logic, package count is reduced by 1/3.

lines, and in permanent configurations, its own IC socket. Some of the devices on each package will be unused. That is, there is a certain amount of *overhead* per package and even potential waste of unused devices!

The main goal of NAND/NAND logic is to

reduce the waste and overhead of such *mixed gate* implementations. Figure 4-11B shows how this is done for this particular function. This circuit performs the same function as the one in Fig. 4-11A, and while the *device count* is greater (six instead of five) the *package count* is less (two instead of three).

The savings in package count comes to one-third. Percentage-wise this is quite significant. And if you are manufacturing hundreds or thousands of such circuits for commercial distribution, the savings would be impressive on an actual dollar basis. Even as an experimenter/hobbyist, however, reducing package count and associated overhead is a very desirable goal.

Another example of NAND/NAND logic is given in Fig. 4-12. If you implement Equation 4-11 literally—term for term and gate for gate—you will

need a total of four IC packages. This is because each device in the circuit of Fig. 4-12A is different.

$$AB + \overline{\overline{C} + D} = Z \quad 4-11$$

A partial improvement is to use NOR/NOR logic for a portion of the circuit, as shown in Fig. 4-12B. This reduces package count by 50%, as only two IC packages are needed.

Another two-fold reduction is realized through NAND/NAND logic in Fig. 4-12C. One 74LS00

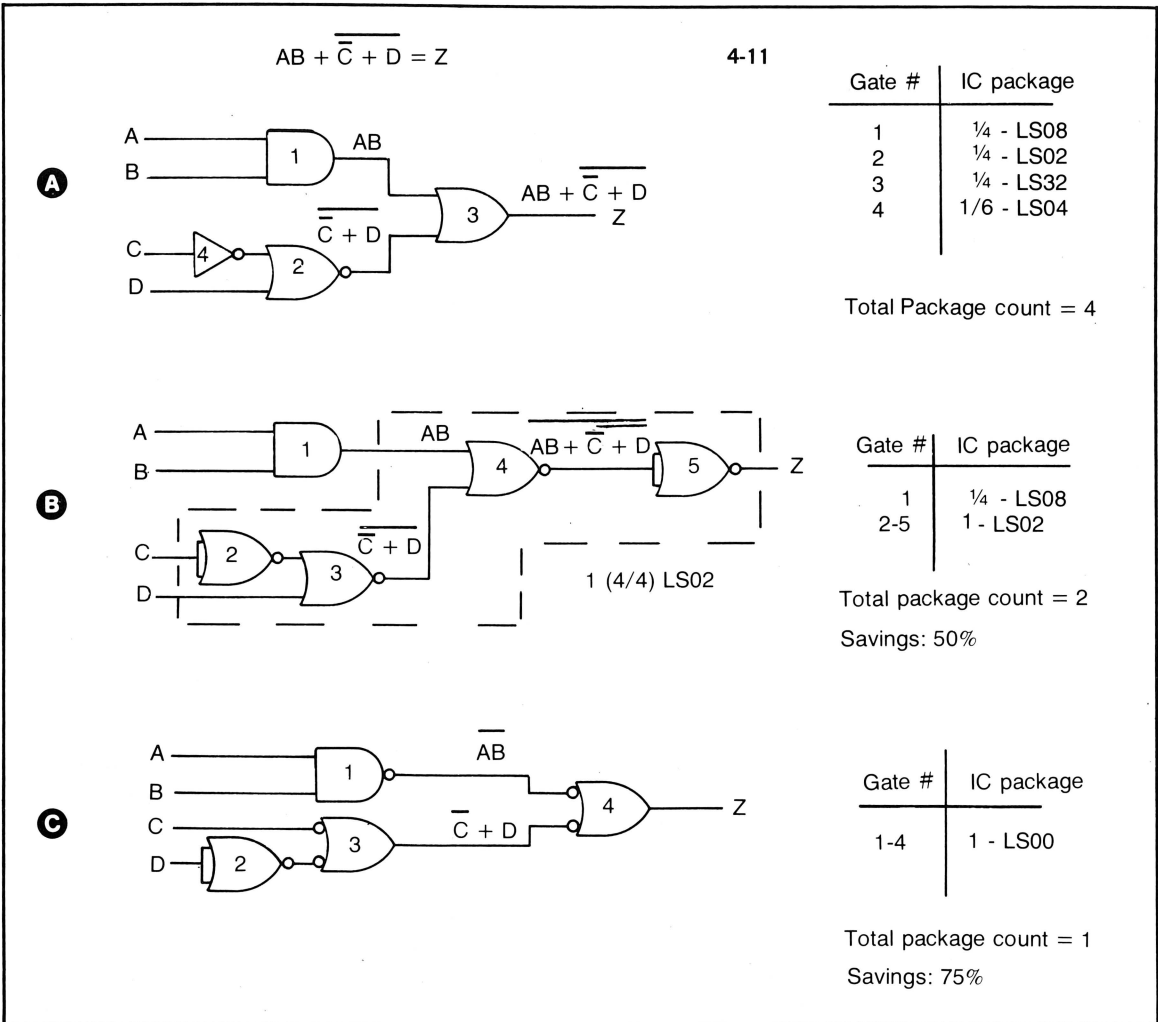


Fig. 4-12. Another example of NAND/NAND Logic. A) Literal schematic. B) Partial reduction using NOR/NOR logic. C) Further reduction with NAND/NAND logic.

quad NAND package does the whole job for a 75% savings in package count compared to the literal implementation of Fig. 4-12A!

Two points about these and similar examples should be stressed. There are cases in which the situation of the last example might be reversed. That is, NOR/NOR logic might result in a lower package count than NAND/NAND. In such in-

stances, you would use the former, as package count is the primary consideration. Also, the method used for the conversion of a literal schematic into a NAND/NAND diagram involves simple inspection in most cases. You should, however, write the generated terms on the signal lines in the schematic to assist you in the conversion and to serve as a double-check later on.

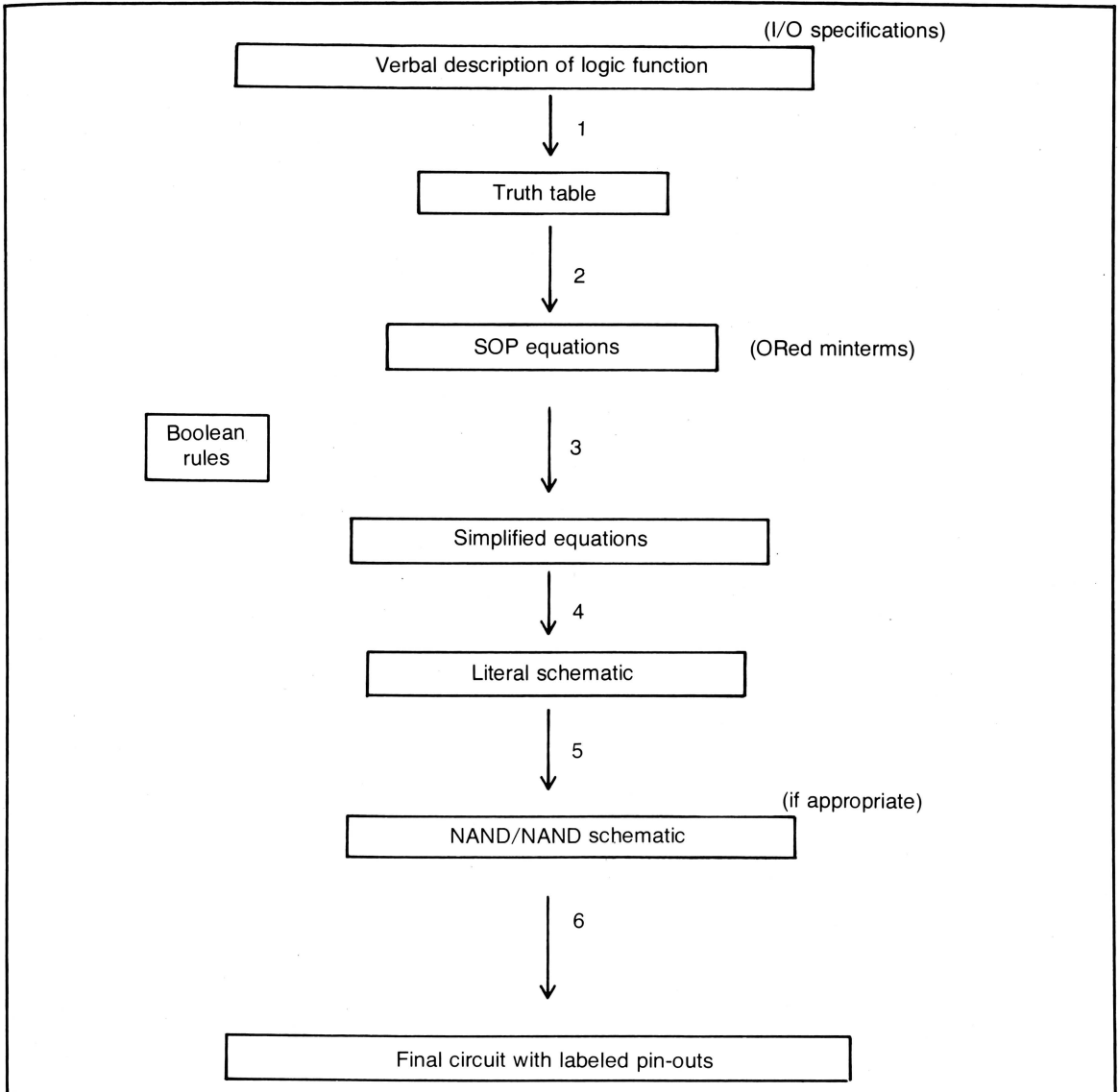


Fig. 4-13. Overview of a typical design sequence.

Now let's turn to some practical design work.

INTRODUCTION TO THE DESIGN EXPERIMENTS

The whole point of this chapter is to present a rational orderly approach to the design of SSI combinational circuits. The concluding experiments that follow are in the form of problem-solution exercises that support the preceding material. Rather than give examples of arbitrary Boolean functions, I have included design problems for small but practical circuits that might be a part of a larger system. Also, the type of functions given are the basis for certain MSI devices discussed later. Both factors may serve as additional motivation for tackling these design experiments.

A general design sequence is presented in Fig. 4-13; it relies on the various methods of Boolean description (truth table, equation, and schematic) and on the rules of simplification already covered.

When given a design problem, you should read the verbal problem statement carefully. Always think in terms of how many inputs and outputs are involved and what these inputs and outputs should be. The first step, shown in the figure, is to translate this verbal statement into a truth table. This is the most crucial step in the process. If you err here, the mistake will be carried through all the other steps. If you've interpreted the problem statement correctly, the subsequent steps will follow easily as they are fairly mechanical.

From the truth table you write an SOP (ORed minterms) expression for each output column. As you have seen, simplification may or may not be indicated, depending on the exact function(s) to be implemented. Next, a literal schematic is drawn with all lines labeled with the correct terms for *housekeeping* purposes. Then a NAND/NAND schematic is drawn, and finally the pin-outs (power, ground, and all signal lines) are numbered.

Wire the circuit on the solderless breadboard, then run and configure BDIS for the right number of circuit input (annunciator) and output (pushbutton) columns.

Generate the truth table on the screen and check it against the design truth table from step 1.

Disagreement between the two means an error somewhere in steps 2 to 6 or in the wiring. Check the wiring first, as this is where many errors occur. If the wiring is right, then you may have made an error in translating the truth table in subsequent steps.

It is possible to have agreement between the design and generated truth tables and still be wrong, however! That is, you could have followed all the mechanical steps correctly from the design truth table onward but may still have misinterpreted the problem statement. This is why the first step is so critical.

Keep in mind that the circuits presented are probably as complex as you would normally construct using SSI devices exclusively. More sophisticated functions are implemented on MSI packages, a subject covered later. Also, the only step unique to this section is the use of NAND/NAND logic; the overall design sequence is otherwise generally applicable. If you are able to solve, or make a reasonable attempt at solving, the problems that follow, you should feel confident in having acquired basic and very useful design skills.

EXPERIMENT 7, SSI DESIGN I

Problem Statement

Design a circuit that indicates whether two signals are the same (both high or low) or different. The single output line of the 2-input circuit should register high if the inputs are different and low if they are the same. Use NAND/NAND logic.

Hint: There are two solutions to this problem. The easier one requires five NAND gates, the more difficult one only four.

Partial Solution

1. Translate the verbal statement into a 2-input/1-output column truth table, Fig. 4-14A. Note how the truth table meets the design specifications given above.

2. The minterms for the two high states are combined into an SOP high type expression and equated to the output, in Fig. 4-14B.

A

Line #	B	A	X
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

$$\begin{aligned} \longrightarrow m1 &= \bar{A}B = X = 1 \\ \longrightarrow m2 &= A\bar{B} = X = 1 \end{aligned}$$

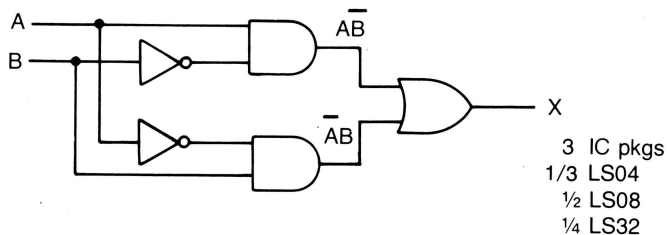
B

$$X = m1 + m2$$

$$X = \bar{A}B + A\bar{B}$$

4-12

C



D

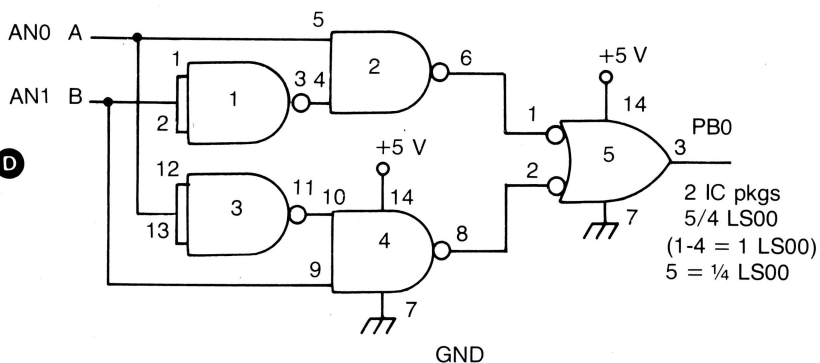


Fig. 4-14. Partial solution for Experiment 7. A) through D) represent the main steps in design of this exclusive-OR (XOR) function, using five NAND gates.

3. The simplification step is bypassed in this partial solution. Instead, the circuit diagram is drawn directly from the equation in literal form, Fig. 4-14C. Three different IC packages are needed in this 5-gate arrangement.

4. Using NAND devices exclusively, you will still require five devices, as in Fig. 4-14D. However, this comes to 2 NAND IC packages, a savings of one-third over the mixed gate circuit of step 3.

5. After numbering and labeling the lines, construct the circuit on the solderless breadboard. Don't forget: ground then power hookup is done first. Note that the annunciator and pushbutton connections are indicated in the small boxes next to the input and output lines of the circuit. Configure BDIS into a 2-in/1-out format. Verify that the 4-line truth table you generate corresponds to the truth table of Fig. 4-14A.

Complete Solution

6. The equation for this circuit, Equation 4-12, was derived from the truth table of Fig. 4-14A. It looks compact enough but it can be simplified:

$$\begin{aligned}
 X &= \overline{A}\overline{B} + \overline{A}B & 4-12 \\
 &= \overline{A}\overline{A} + \overline{A}\overline{B} + \overline{A}B + \overline{A}B & \text{Identity/Rule 3 (two} \\
 & & \text{zeros are added)} \\
 &= \overline{A}(\overline{A} + \overline{B}) + \overline{A}(B + B) & \text{Distributive/Rule 8} \\
 &= \overline{A}(\overline{A}\overline{B}) + \overline{A}(AB) & \text{De Morgan/Rule 10} \\
 &= \overline{A}\overline{B}(A+B) & \text{Distributive/Rule 8}
 \end{aligned}$$

The final expression I will call Equation 4-13. The input variables are on the right, and the output variable on the left. This is a reversal of the informal convention.

$$X = \overline{A}(\overline{A}\overline{B}) + \overline{A}(AB) = \overline{A}\overline{B}(A+B) \quad 4-13$$

7. By adding zeros to the original equation, $\overline{A}\overline{A}$ and $\overline{B}\overline{B}$, the value of the expression is unchanged. This allowed the factoring out of the common term, to give Equation 4-13. This may not look simpler than

Equation 4-12, however, if you examine the two equations you will see that there are fewer Boolean operations in the latter equation.

8. Implement Equation 4-13, using NAND/NAND symbology as in Fig. 4-15B. One less gate is used, as you can see from comparing this circuit with Fig. 4-14D. This results in a 50% savings in package count, an improvement over the partial solution above.

9. Build the circuit from your pin-out NAND/NAND diagram. Fig. 4-15B. Confirm its operation by generating the truth table using BDIS, as before.

Discussion

This was not an easy problem, so if you did not see your way through to the complete solution, do not feel badly. Hopefully, you were able to arrive at the partial solution without much difficulty.

One thing stressed in this experiment is that you should always look for alternative solutions if you have any suspicion that your equation is not totally simplified. Equation 4-12 looked simple, but the trick of adding zero terms did wonders: it cut package count in half by eliminating the one extra, nagging gate. In cases where elimination of one device can save an entire package, look critically at your preliminary solution to see if it can be improved.

The other feature of this experiment is that it demonstrates another important combinational function: Exclusive-OR, or XOR for short. The logic symbol and the special operator (an encircled + sign) are shown in Fig. 4-15C. The 74LS86 is a commonly available quad XOR IC package. Introduction of this function was delayed so that you could fully appreciate how and why it works using the Boolean descriptive method of this chapter.

XOR devices are also called comparators, because they indicate whether two signals are the same or different. XOR circuits can compare 8-bit words, perform arithmetic functions, and can check for errors in data transmission (parity checking). Strictly speaking, the simplest XOR package, the LS86, is classified as an MSI device because it uses

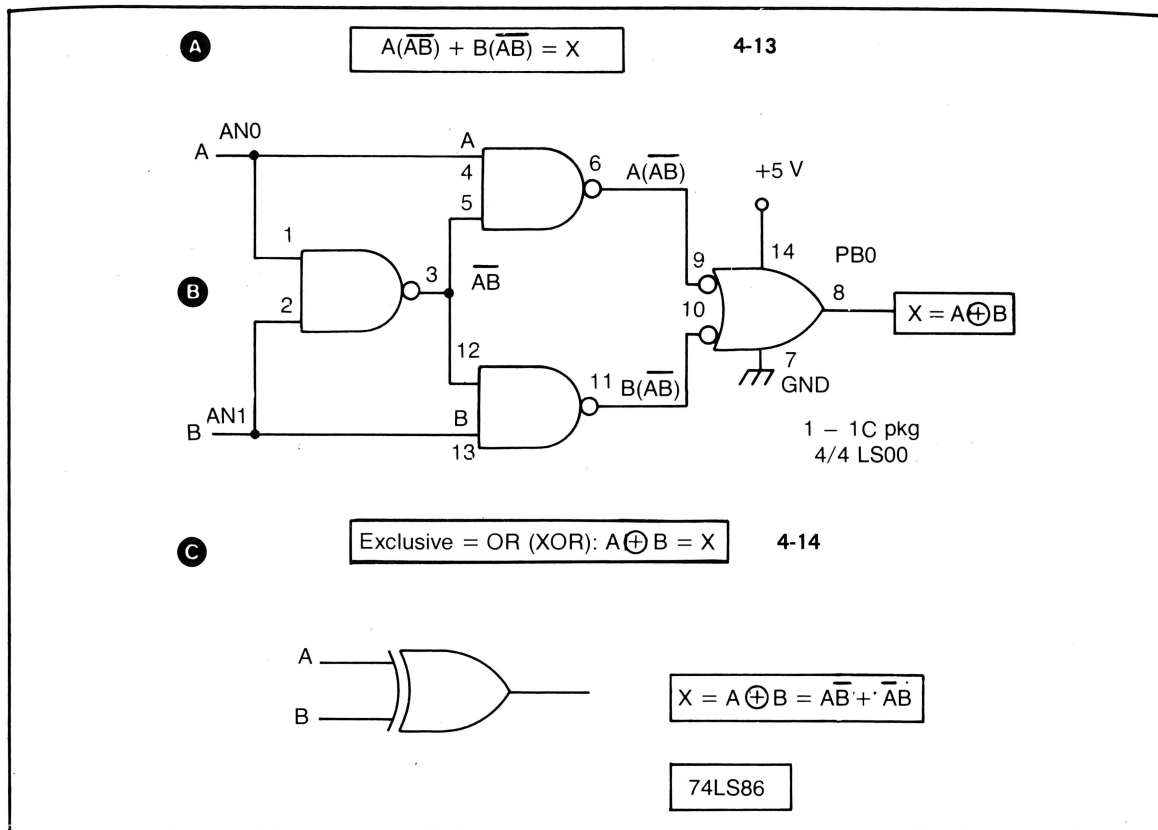


Fig. 4-15. Complete solution for Experiment 7. Simplification to Equation 4-13 (see text in the experiment) has eliminated one device, and thereby an entire package in the final circuit shown in (B). In (C), the special operator and schematic symbol for XOR are shown.

twelve gates per package. I will say more about XOR in the combinational MSI chapter.

EXPERIMENT 8, SSI DESIGN II

Problem Statement

Design a circuit with two inputs and four outputs, in which only one of the four outputs will be low/0 for any one of the input states. Remember that there are four ($2^2 = 4$) possible input states. Figure 4-16 is the general scheme of the design.

Optional experiment: Design another similar circuit in which only one of the four outputs will be high/1 for any one of the input state.

Hint: the first design employs NAND/NAND logic. The second employs NOR/NOR logic.

Solution

1. Construct the truth table for the four input states. This table will have four output columns; each output variable will be low/0 for only one input state. Conversely, you would say that for any given input state, only one output will be low/0, the rest will be high/1. See Fig. 4-17A. Write the minterms for each output low on the appropriate line.

2. From the truth table, write the four equations. See Fig. 4-17B. Each consists of one minterm, and is equated to low/0, the value of its output variable. These equations are of the SOP, low form. You are specifying the sole condition when the output is low in each case. For example, output W is 0 only when both A and B are 0. Continue for X, Y, and Z.

3. The four, single minterm equations are rewritten

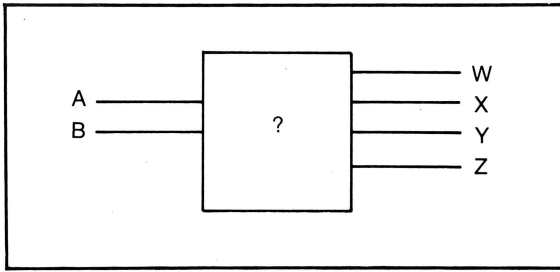


Fig. 4-16. General input/output scheme for the design problem of Experiment 8.

ten for a high/1/true output condition, as indicated to the right of the four original equations in Fig. 4-17B. These are SOP high equations. Actually, this conversion to SOP high form is not absolutely necessary, as one could draw the schematic from the SOP low set of equations to the left; however the resulting gate count is less if you use the SOP high set to the right.

4. Draw the schematic using NAND/NAND sym-

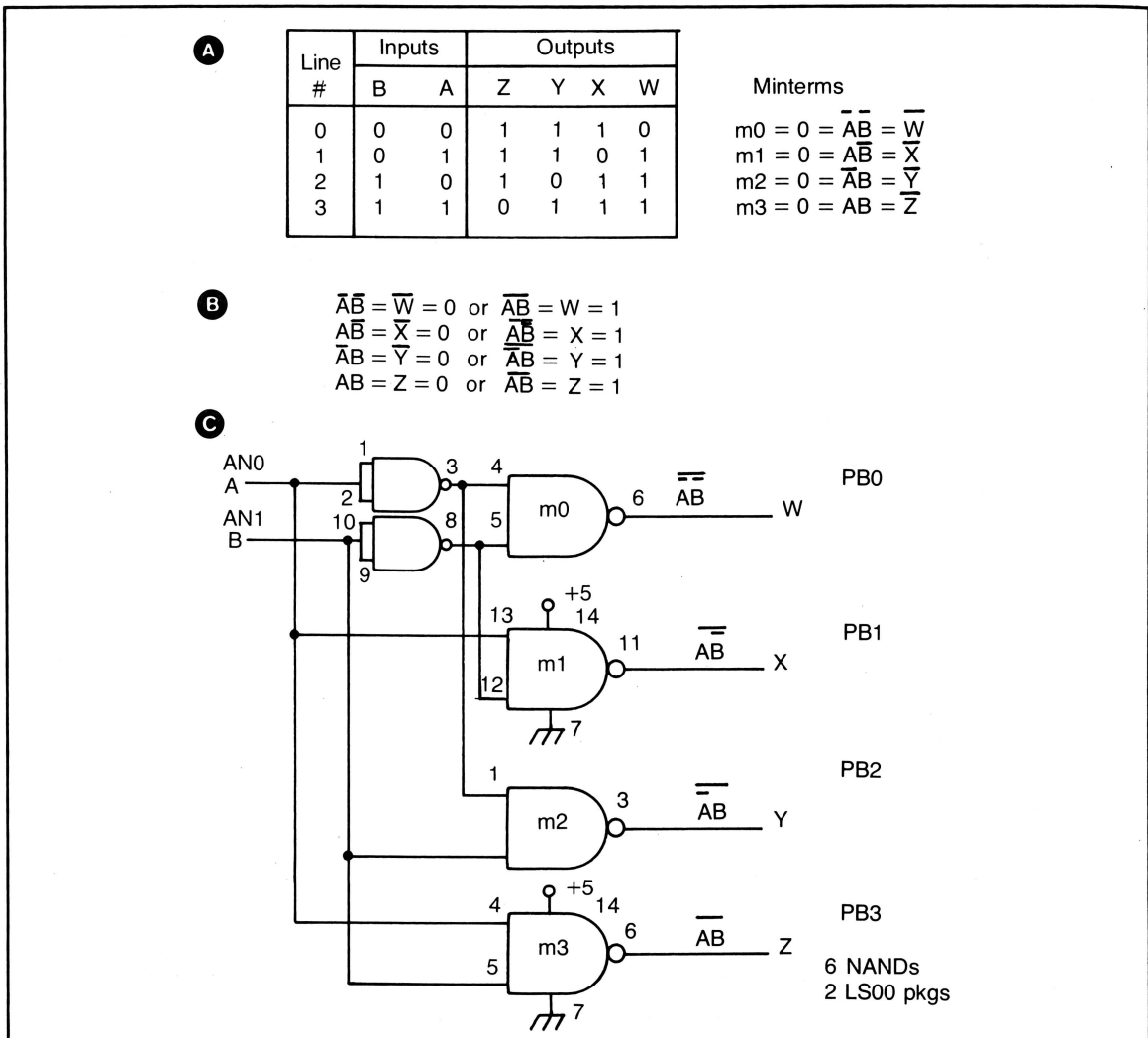


Fig. 4-17. Solution for the 2-to-4 line decoder of Experiment 8. The outputs are active-low. The truth table, equations and final NAND/NAND logic circuit are discussed in the text.

bology. Label the pins appropriately for the signal names, terms generated, and annunciator and pushbutton connections. See Fig. 4-17C.

5. Hook up the circuit, run DBIS and configure to a two-input/four-output column format. Verify circuit operation by generating the 4-line truth table.

6. As an optional exercise, try implementing the other design mentioned in the problem statement, namely, active high/1 output states instead of low/0 states. The truth table, equation set and labeled NOR/NOR schematic are shown in Fig. 4-18. The procedure is similar to that for the NAND/NAND implementation.

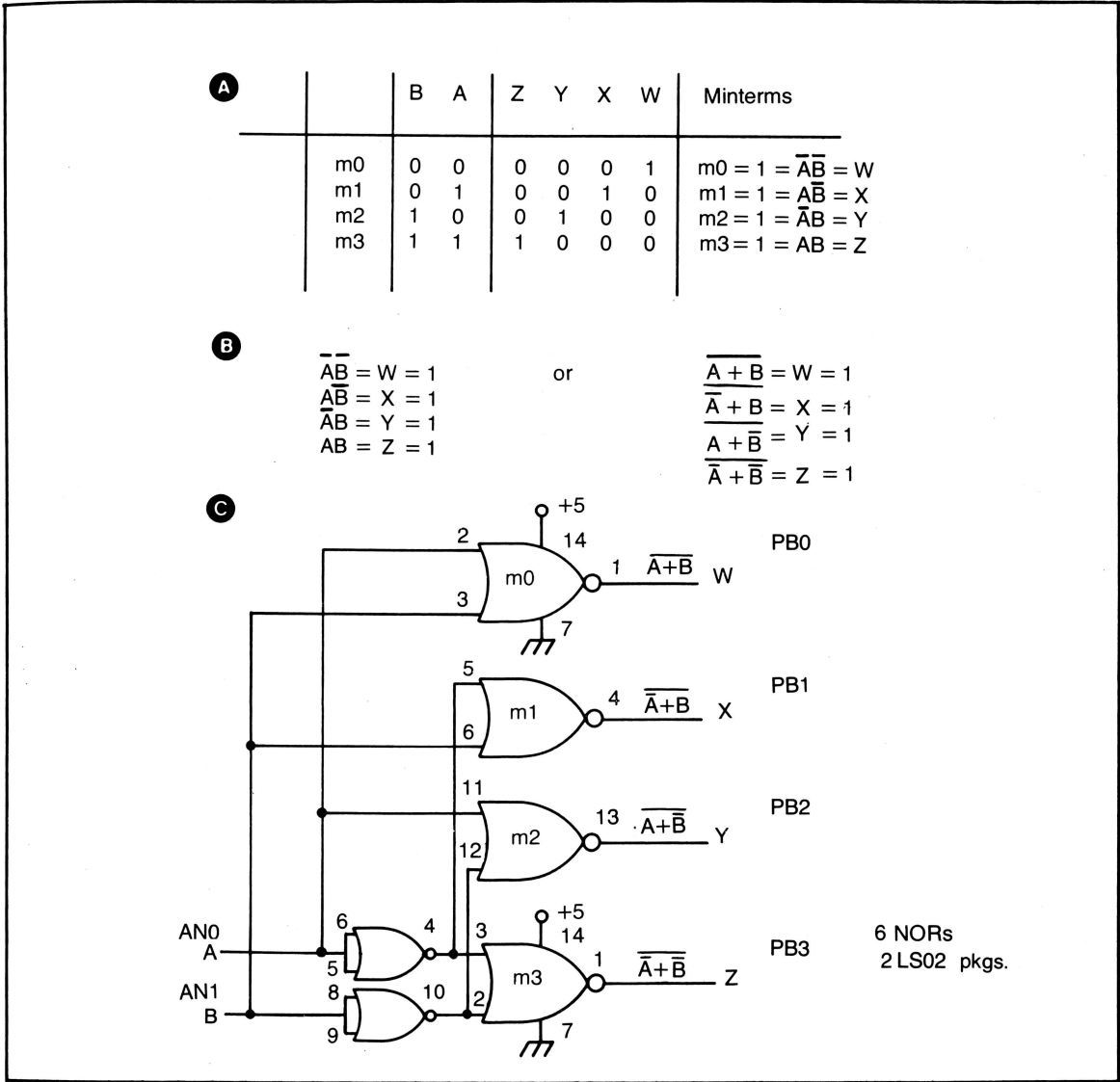


Fig. 4-18. Alternate solution to the 2-to-4 line decoder problem, using NOR/NOR Logic. The outputs are active-high. Economy of package count is the same as in the NAND/NAND solution. The choice between the two depends on the particular application.

Discussion

Before discussing the actual details of design, you may be wondering exactly what the circuits of Figs. 4-17 and 4-18 actually do. If outputs W, X, Y and Z were relabeled as 0, 1, 2, and 3 respectively, then you can see that these circuits actually *decode* a 2-bit binary number into a decimal number 0-3! In a sense, these circuits could be described as simple binary-to-decimal code converters.

In this circuit, there are two input lines, or four possible input states. In a similar circuit with three input lines, there would be 2^3 or 8 possible input states; in a 4-input circuit, there would be 2^4 or 16 possible states. In each case there would be eight and sixteen output lines, respectively. The terminology for these circuits is 2-to-4 line, 3-to-8 line, and 4-to-16 line decoder, respectively.

There are other interpretations for this decoding function, such as device activation, address decoding, data distribution and so on. These functions are found on the more complex MSI decoder/multiplexer chips that we will discuss later. For the moment, let's look at the two implementations of the 2-to-4 line decoder in Fig. 4-17 and 4-18.

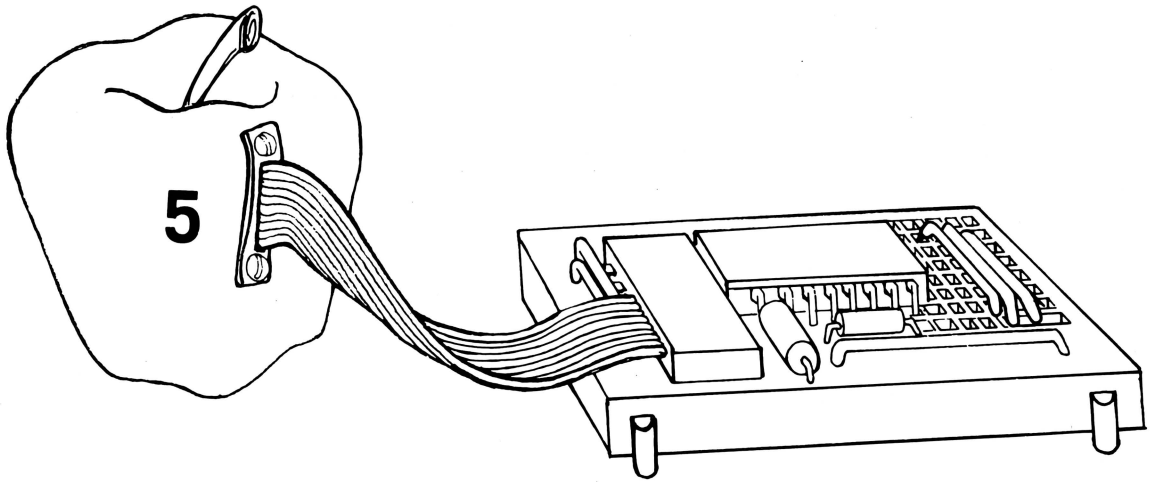
In the NAND/NAND portion of this experiment, the important thing was to remember how to write an equation for an output low/0 state. It involved nothing more than taking the input minterm for the output low state and setting it equal to zero. You are in effect specifying when the input state is false/low. Then, by taking the inverse of both sides

of equation, you are stating when the output is true/high. Again, by example:

$$\begin{aligned}\overline{AB} &= \overline{W} = 0 \\ &\text{or} \\ \overline{\overline{AB}} &= W = 1\end{aligned}$$

Both equations are valid. Again, the only reason we want to work with the second, SOP high form of expression is that the resultant circuit is a bit simpler. Note that in this circuit, we cannot utilize the output of one equation as part of the input for another, as was done in majority detector example (Fig. 4-9). But we can use the inverted input lines for A and B twice, thereby realizing some small savings.

Besides the active low output circuit of Fig. 4-17, there is an alternative active high output circuit performing the same function, shown in Fig. 4-18. If you compare the truth tables for the two circuits, you'll note that they express the same general function of a 2-to-4 line decoder. The difference is that in the second, NOR/NOR, circuit, each output is high/1 for any given input state, and low/0 otherwise. That is, the decoder outputs are *active-high*, rather than active-low. In certain applications one level of output might be preferred over the other, however, there is no implicit advantage in having active-low or active-high outputs. Certainly, the gate and chip count of the two circuits are the same, and they are therefore equally efficient from the standpoint of construction.



Discrete Electronic Components and Laws

So far you've learned about the building blocks of digital circuits: the combinational logic devices and the Boolean Laws that govern them. You have learned not only about the specific tasks they perform, but also how these devices can be combined into working circuits, that is, the rudiments of digital design. Most of this material has been on a *functional* level. Using the imagery introduced in the introductory chapter, we have been concerned with what these black boxes do and how to connect them into bigger black boxes (circuits) that will perform more complex functions.

This functional approach can take you a long way in understanding digital devices, to be sure. However, the functional approach isn't enough. You should know something about the electronic components that make up digital ICs, and the laws that govern them.

A fair amount of space is devoted to basic electronics at this point for several reasons. Obviously, you must know about the innards of digital ICs if you are to use them properly. By detailing the components, you'll learn about basic current and

voltage relationships within circuits, about the workings of semiconductors, and about the meaning of digital IC ratings and specifications.

In short, you are going to descend a few levels into the black box and look inside. This is the world of discrete components: resistors, capacitors, diodes and transistors. Each component has its own law or operating principle which can be understood by means of a graph, simple equation or mechanical analogy. In this chapter, we'll look at two passive components first, the resistor and capacitor. *Passive* means that they modify signals but do not amplify them. By doing so we'll be covering a necessary kernel of basic electronics, and be laying the groundwork for understanding diode and transistor action and TTL circuit operation, which is detailed in Chapter 6.

THE RESISTOR AND ASSOCIATED LAWS

This section discusses voltage, current, Ohm's Law, and Kirchhoff's Laws. It also includes important information on resistor values, ratings and uses.

Voltage and Current

Voltage and current can be easily understood by means of a common analogy. A closed hydraulic system consisting of water-filled tubing and a pump is shown in Fig. 5-1. The water passes through the tube at a certain *flow rate* given in gallons/minute or in similar units. This flow occurs against a certain frictional resistance within the tubing, therefore some pumping force is necessary.

Under the influence of the pump, water moves in the direction indicated, namely from a region of high to low pressure. It flows down a *pressure differential* or drop; the pressure at point A, P_A , is higher than that at point B, P_B . If there were no differential, water would not flow at all, and if the differential were reversed ($B > A$) then it would flow in the reverse direction. One can say that there is a *pressure drop* over this segment of tubing [$P_A - P_B$] which represents the pressure lost in overcoming the frictional resistance of the tubing.

When a small unit of water arrives back at the intake side of the pump after its travels around the circuit, it is raised to a higher pressure region of the system by the action of the pump. This is the

meaning of the + and - signs in the figure. Hydraulic pressure can be expressed in units of pounds/sq.in., atmosphere, mm of mercury (Hg) and so on.

Voltage and current are almost exactly analogous to the hydraulic quantities given above. Historically, electricity was thought to be a type of fluid which possesses *charge* and which could move from one place to another under the influence of an electrical pressure or *voltage*. By convention the charge of electricity was taken to be *positive*.

As Fig 5-2A illustrates, a voltage source pushes this positive electrical fluid through a wire. The battery (electrical pump) raises the positive charges from a low (negative) to a high (positive) potential. This positive charge flows out of the positive battery terminal, through the wire and then into the negative terminal to be raised up to a high potential again. Positive current flow is in the direction of (+) to (-). (You could use the image of the positive terminal repelling the positive charges, with the negative terminal attracting them.)

Of course, the major charge carriers in wire are really negatively charged electrons, but this

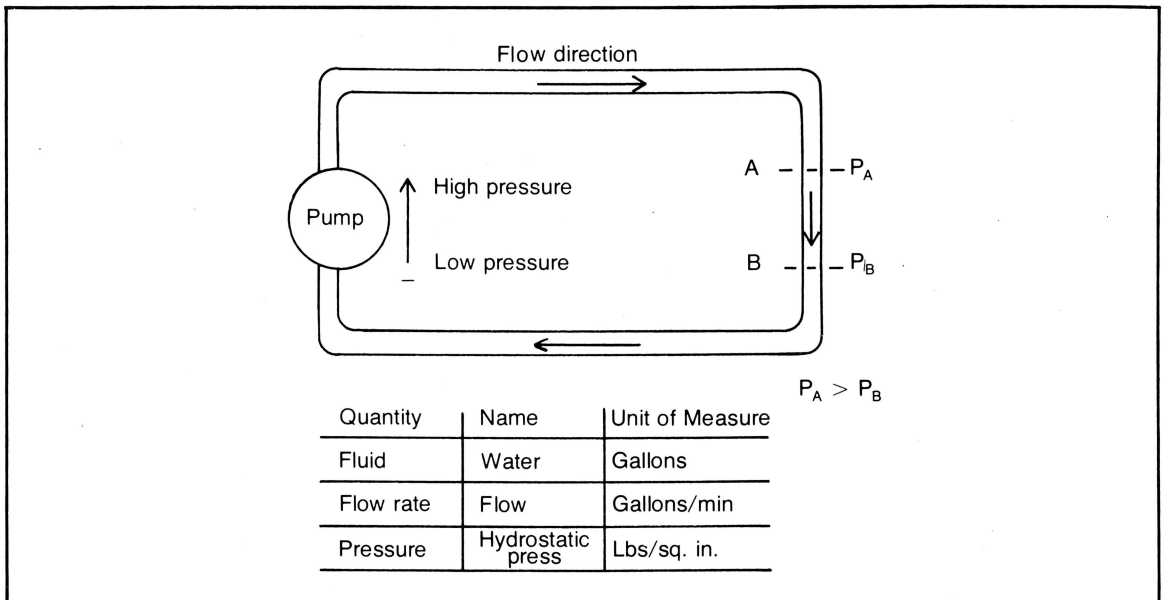
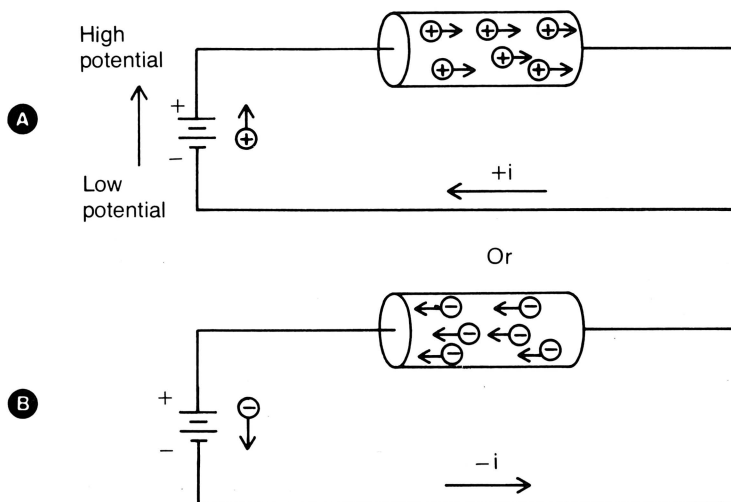


Fig. 5-1. Hydraulic circuit, with pump, water flow and pipe resistance analogous to the electrical quantities of voltage, current and electrical resistance.



Quantity	Name	Unit of Measure
Fluid	Charge	Coulomb = 6.24×10^{18} unit charge
Flow rate	Current	Ampere = 1 coul/sec
Pressure	Electric potential	Volt = unit of potential work

Fig. 5-2. Current conventions. Electrical current flow in a wire can be represented by positive charge (A), or as negative charge flow (B). Units of measure are given in the table at the bottom of the figure.

was not appreciated when the positive current convention was established some two centuries ago. This physically more accurate representation of electron flow is given in Fig. 5-2B. Here the directions are reversed, with negative current flow ($-i$) proceeding from the negative to the positive terminal. Rather than do the mental gymnastics of thinking of negative voltage as a high pressure relative to negative charges, do something easier. Use the like-repels-like idea just mentioned; negative current is repelled by the negative terminal and attracted to the positive one.

Thus, for any case of current flow in a wire, you can speak of two currents flowing at the same time, but in opposite directions. It is necessary to choose either the $(+i)$ or $(-i)$ convention and stick to it.

Frankly, the historical image of positive fluid driven by a positive electrical voltage and moving from high to low electrical potential (voltage drop across a resistance) is more physically appealing. The positive current flow convention is still very much in use, is preferred in engineering texts over the negative current flow convention, and is the convention used in this book. (You will come upon both representations for direction of current flow in your readings with the nature of current unspecified. Just remember the above conventions and the meaning will be clear.)

To continue along the lines of the hydraulic analogy, electrical charge can also be quantified, like any other fluid. A physical unit, the *coulomb*, consists of a large number of very small unit charges $(+ \text{ or } -)$. Electrical flow rate or *current* is

expressed in *amperes*. One ampere is one coulomb passing a point in a wire each second. Let's formalize this a bit.

If we call the quantity of charge Q , then the amount of charge, ΔQ , passing a point per unit of time, Δt , is the current:

$$I = \Delta Q / \Delta t \quad 5-1a$$

The delta symbol means the difference or change in the parameter it proceeds. Replacing the delta symbol with the letter d , we have

$$I = dQ/dt \quad 5-1b$$

This equation means the same thing but expresses current more conventionally as a so called *time derivative* of charge: the rate of change (or passage) of charge with respect to time. Generally, the rate of change of any physical parameter with respect to time assumes the form of the time derivative:

$$\text{Rate of Change} = \frac{d \{ \text{Parameter Symbol} \}}{dt}$$

For instance, if x is distance, then dx/dt is velocity. This notation is useful later on, when I explain capacitance.

I can speak of voltage as a sort of electrical pressure which drives charge carriers (+ or -) down a wire. In classic physical terms, the volt is a potential unit of work performed on a quantity of charge a certain distance against an electrical gradient. This would be analogous to moving a weight a certain distance against the force of gravity.

A more useful definition is to think of voltage as a potential to drive current through a conductor *against a resistance*. This is similar to the water pump system, in which the driving pressure is needed to overcome the frictional resistance of the pipe in order for flow to occur.

Ohm's Law

As illustrated in Fig. 5-3A, the resistance to current flow in a simple electrical circuit can be lumped into a quantity R . (Even copper wire has a small but measurable resistance.) Here, the voltage source forces charge carriers (we use the positive current convention) through the wire and

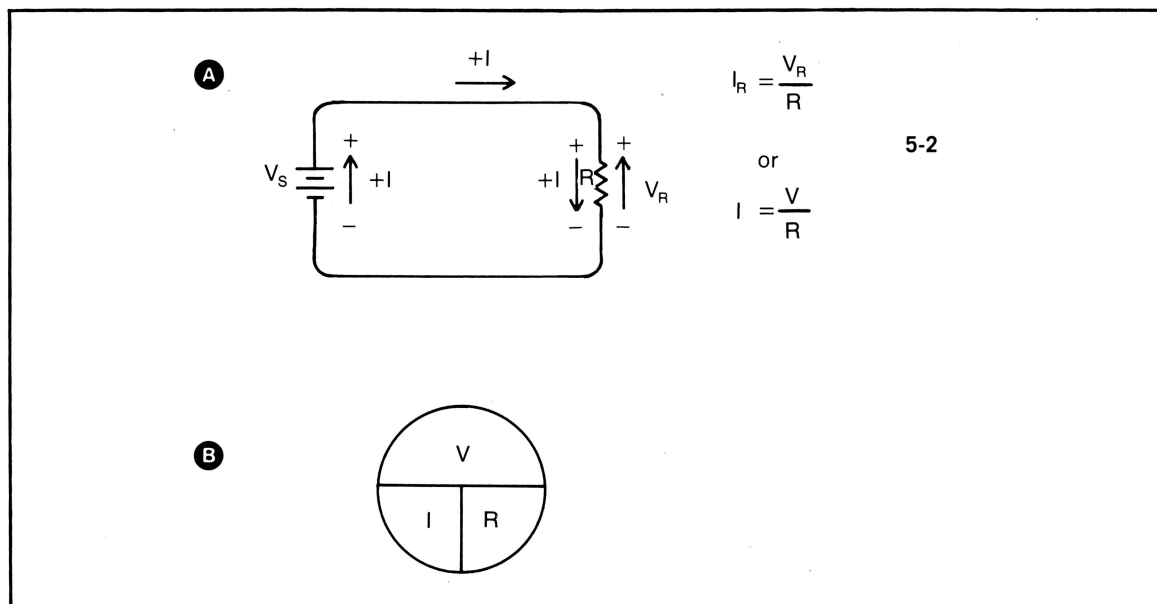


Fig. 5-3. Ohm's Law. (A) the basic relationship for current, voltage and resistance. Note the voltage drop across R and its polarity, positive current falls down the potential difference across the resistor. (B) A simple mnemonic for Ohm's Law.

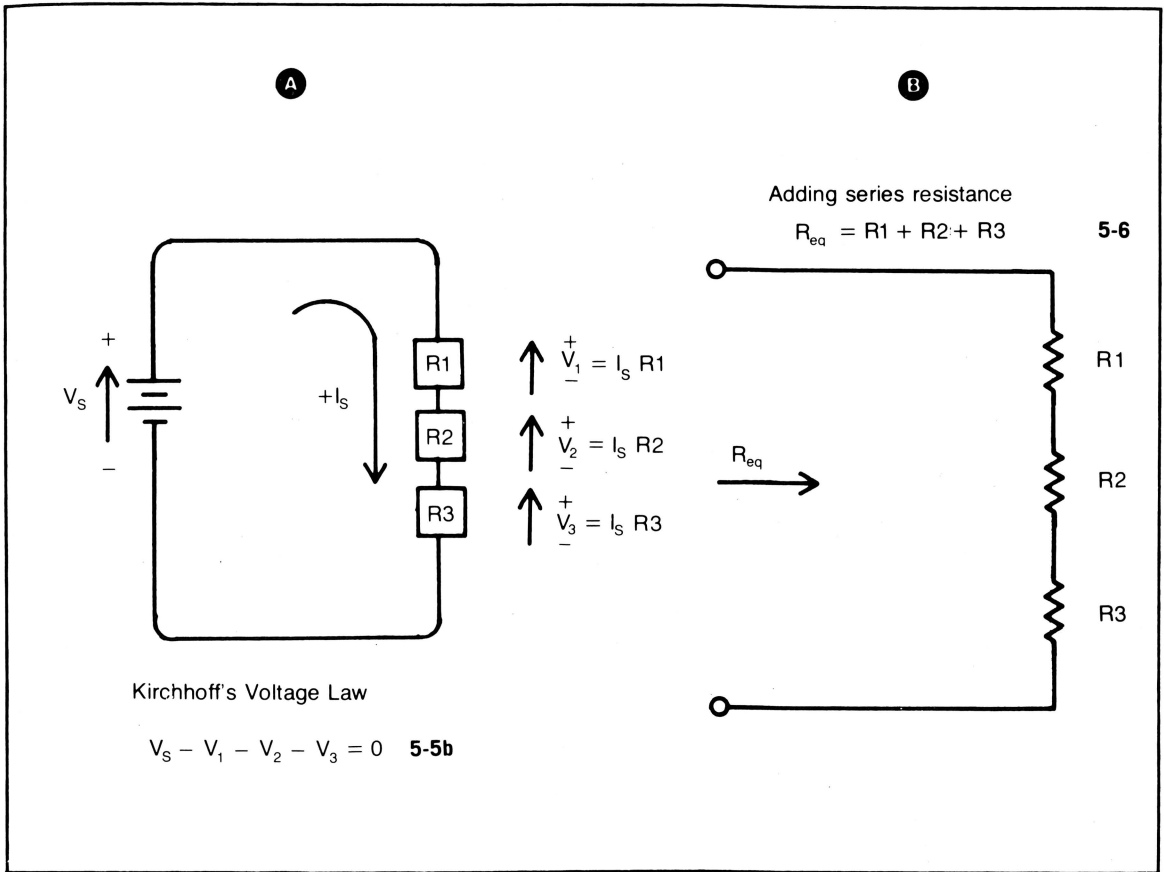


Fig. 5-4. (A) Kirchhoff's Voltage Law shows how to add voltages around a loop. Source voltage minus voltage drops equals zero. (B) It also is the basis for proving the linear addition of series resistors.

against this lumped resistance. Resistance in electrical circuits is measured in *Ohms*. We can relate current, voltage, and resistance by means of Ohm's Law:

$$I = V/R \quad \mathbf{5-2}$$

Equation 5-2 states that current flow is proportional to voltage (driving pressure) and inversely proportional to resistance. This makes physical sense. Using the units of measure just mentioned, the equation states that, "It takes one volt of driving potential to create one ampere of current flow against a resistance of one ohm."

For a 9 V source and a 90 ohm resistor, total current flow in this circuit would be $9 \text{ V}/90 \text{ ohms} =$

1/10 ampere (A) or 100 milliamperes (mA).

A little algebra will allow you to convert Equation 5-2 into the other forms of Ohm's Law.

$$V = IR \quad \mathbf{5-3}$$

$$R = V/I \quad \mathbf{5-4}$$

An interpretation of Equation 5-3 goes like this: given the value of a resistor and the current flowing through it, you can find the voltage drop (analogous to pressure differential or gradient) across it. An interpretation of Equation 5-4 goes like this: in order to limit current flow to a given value under a certain specified voltage, use a resistor with value R .

Figure 5-3B is a memory aid for Ohm's Law.

Point to the quantity desired and it gives you the correct expression. Naturally, this aid should not be necessary if you remember the physical sense of the law: a voltage potential driving a current through a resistance.

Ohm's Law has a number of practical consequences. You can use it to do some common sense analysis of more complex circuits than the one in Fig. 5-3, and to determine component values for circuits you may be constructing.

Kirchhoff's Voltage Law

Kirchhoff's Voltage Law states that the sum of voltages around a *loop* are equal to zero. This law applies to *series* circuits such as the one in Fig. 5-4A. The voltage source and the three resistances are in series, that is, end to end. The source current (I_s) flowing through each of these resistances is the same because the current path is continuous. As the current passes through each resistance it loses some of its driving force, in much the same way as water loses some of its forward driving pressure in passing through a series of pipe sections (hydraulic resistance elements). Just as there is a pressure drop across each pipe section, there is a *voltage drop* across each electrical resistance. The sum of these voltage drops equals the source voltage:

$$V_{\text{source}} = V_1 + V_2 + V_3 \quad 5-5a$$

Across each resistance there is a voltage drop. The *polarity* of the voltage drop opposes the polarity of the source voltage. This is only logical, as the voltage is higher on the upstream side of the resistance, and lower on the downstream side. Current would not, after all, flow unless this voltage differential existed! If we take the polarity of the voltage drops relative to V_{source} into consideration, we have:

$$V_{\text{source}} - V_1 - V_2 - V_3 = 0 \quad 5-5b$$

This is just Equation 5-5a rearranged to conform to the verbal statement of Kirchhoff's Voltage Law.

Lest you think otherwise, the voltage drop

across each resistance is not an abstract notion, but a real, physically measurable quantity. If you built the circuit in Fig. 5-4A you could measure a separate voltage across each of the resistances, the sum of which would approximate V_{source} , within the limits of measurement error.

One practical result of Ohm's Law and Kirchhoff's Voltage Law is that you can see how to add series resistances. Not only that, you can prove this series addition relationship.

$$\begin{aligned} V_s &= V_1 + V_2 + V_3 && \text{Kirchhoff's Voltage Law} \\ &= R_1 I_s + R_2 I_s + R_3 I_s && \text{Ohm's Law} \\ &= I_s (R_1 + R_2 + R_3) \\ &= I_s R_{\text{eq}} \end{aligned}$$

where R_{eq} is the equivalent resistance.

$$R_{\text{eq}} = R_1 + R_2 + R_3 \quad \text{Series Addition} \quad 5-6$$

Remember, it is the fact that the current is the same through all segments of the loop that allows you to factor out I_{source} in the third step. The total or equivalent resistance is then "seen" from the viewpoint of the voltage source as the sum of the individual resistors, as in Fig. 5-4B.

Another practical application of Ohm's Law and Kirchhoff's Voltage Law is that of *voltage division*. Figure 5-5 is a redraw version of Fig. 5-4. It shows that the voltage drop across one or more series resistors is merely the *ratio* of that resistance to the total resistance, times the source voltage. This should make intuitive sense, and the relationship could be proven for each case. Let's work out the drop across R_3 (V_{cd} or V_3) as an example. Note that the drop across the total series resistance (V_{ad}) is the same as V_s , and that the current through R_3 (I_3) is the same as I_s .

$$\begin{aligned} I_3 &= V / R_3 = V_{\text{ad}} / R_{\text{eq}} = I_s \\ V_{\text{cd}} &= V_{\text{ad}} (R_3 / R_{\text{eq}}) \\ \text{or} \\ V_3 &= (R_3 / R_{\text{eq}}) V_s \end{aligned}$$

In the figure, a total of five voltage drops are calculated: three for each of the resistors, and two

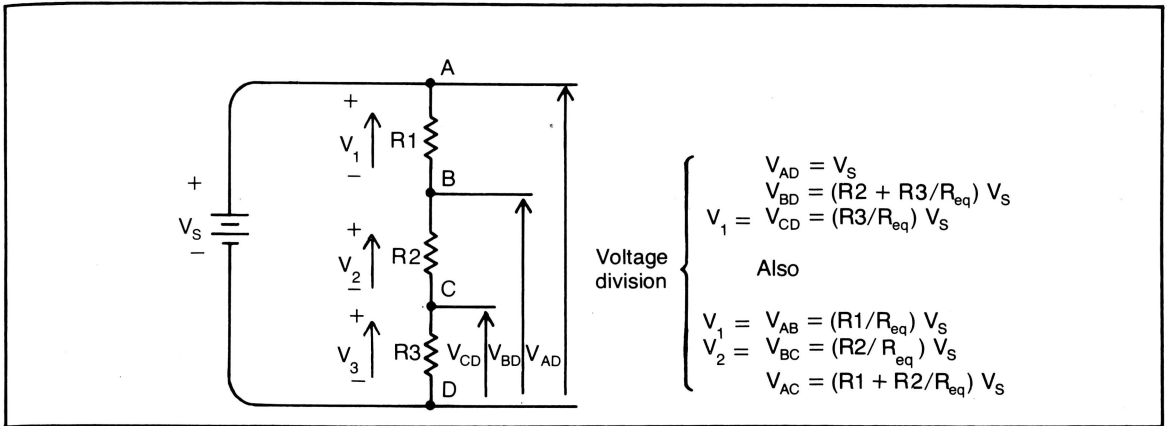


Fig. 5-5. Voltage division among series resistances is a matter of direct proportion or ratios. See text. The various voltage drops which could be measured in this simple three-resistor circuit are shown, with the calculations for each.

for two pairs of resistors. These voltage drops are indicated by the set of arrows between the labeled points in the figure. In short, the voltage drop in series resistance circuits is just an arithmetic proportion or ratio of one or more resistances to the total resistance, R_{eq} .

Kirchhoff's Current Law

Kirchhoff's Current Law states that the sum of currents entering or leaving a point in an electrical circuit is zero. This law applies to *parallel* circuits, like the one in Fig. 5-6. The three resistors are in parallel. This time, it is the *voltage* across each resistor which is the same, and the *current* through each which is divided.

According to the law, current is conserved at any point in the circuit. At point A in Fig. 5-6A, the amount of current entering the point, I_{source} , is equal to the amount leaving it. I_{source} equals the sum of three smaller currents. I_1 , I_2 and I_3 . A similar statement can be made about point B. The equations expressing this current law are:

$$I_{source} = I_1 + I_2 + I_3 \quad 5-7a$$

$$I_{source} - I_1 - I_2 - I_3 = 0 \quad 5-7b$$

Ohm's Law and Kirchhoff's Current Law can be used to show how parallel resistances are added and how current is divided among parallel resistances.

To add resistances in parallel, note that the voltage across all the resistors, V_{AB} , is the same; it is equal to the source voltage, V_S . Using Equation 5-7a and Ohm's Law, we have:

$$\begin{aligned} I_S &= I_1 + I_2 + I_3 \\ &= V_1/R_1 + V_2/R_2 + V_3/R_3 \end{aligned}$$

and since V_1 , V_2 , and $V_3 = V_S$ we have

$$\begin{aligned} I_S &= V_S(1/R_1 + 1/R_2 + 1/R_3) \\ &= V_S(1/R_{eq}) \end{aligned}$$

Rearranging gives Equation 5-8 for parallel addition of resistors.

$$1/R_{eq} = 1/R_1 + 1/R_2 + 1/R_3 \quad 5-8a$$

The equivalent parallel resistance is in reciprocal form in this equation. You must take the sum of the reciprocals of the individual resistors, and take the inverse of the sum to get R_{eq} . A pocket calculator helps. But for any two parallel resistors, use the following, somewhat simpler equation:

$$R_{eq} = \frac{R_1 \times R_2}{R_1 + R_2} \quad 5-8b$$

For two parallel resistors, R_{eq} is their product divided by their sum. In general, the R_{eq} of parallel

resistances is *less* than the *smallest* resistor in the configuration, whatever the number of resistors involved. For a 2-resistor circuit having 10 ohms in parallel with 100 ohms, R_{eq} is found to be 9.1 ohms.

$$R_{eq} = 10 \times 100 / 110 = 9.1 \text{ ohms}$$

Also, for a 2-resistor network, if the resistors are equal, the R_{eq} will be exactly one-half the value of either of them. For two 50 ohm resistors in parallel Equation 5-8b yields 25 ohms.

$$R_{eq} = 50 \times 50 / 100 = 25 \text{ ohms}$$

Likewise, three equal resistors in parallel would have an R_{eq} of one-third the value of any one of them. Four equal resistors have an R_{eq} of one-fourth the value of one of them.

Current is divided among parallel resistances as you would expect. More current flows into the

smaller resistances, less current into the larger resistances. Figure 5-7 is a redrawn version of Fig. 5-6. Keeping in mind that the voltage, V_s , (the voltage between points A and B) is the same for all resistors, we can write for R_1 :

$$\begin{aligned} V_s &= V_1 \\ I_s/R_{eq} &= I_1 \times R_1 && \text{Ohm's Law} \\ I_s(R_{eq}/R_1) &= I_1 && \text{rearranging terms} \\ I_1 &= (R_{eq}/R_1) I_s \end{aligned}$$

That is, the current flowing into any resistor in a parallel configuration is inversely proportional to the ratio of the given resistance to the equivalent resistance. In the example of a 10 ohm in parallel with a 100 ohm resistor, this ratio is $9.1/10 = 0.91$. This means that 91% of the current would flow into the 10 ohm resistor, and only 9% into the 100 ohm resistor!

We can also say that most of the current is

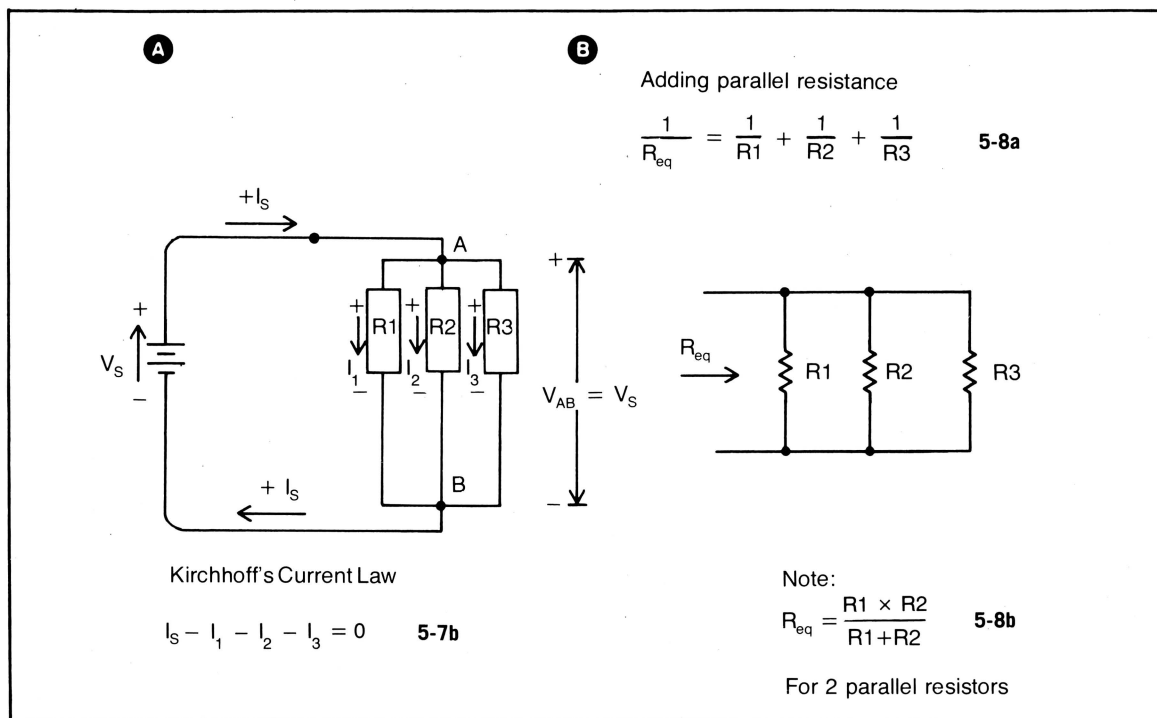


Fig. 5-6. (A) Kirchhoff's Current Law shows how current is divided among parallel branches. The sum of currents entering and leaving a point is equal to zero. (B) It is also the basis for showing how parallel resistances are added, namely, as the sum of their reciprocals.

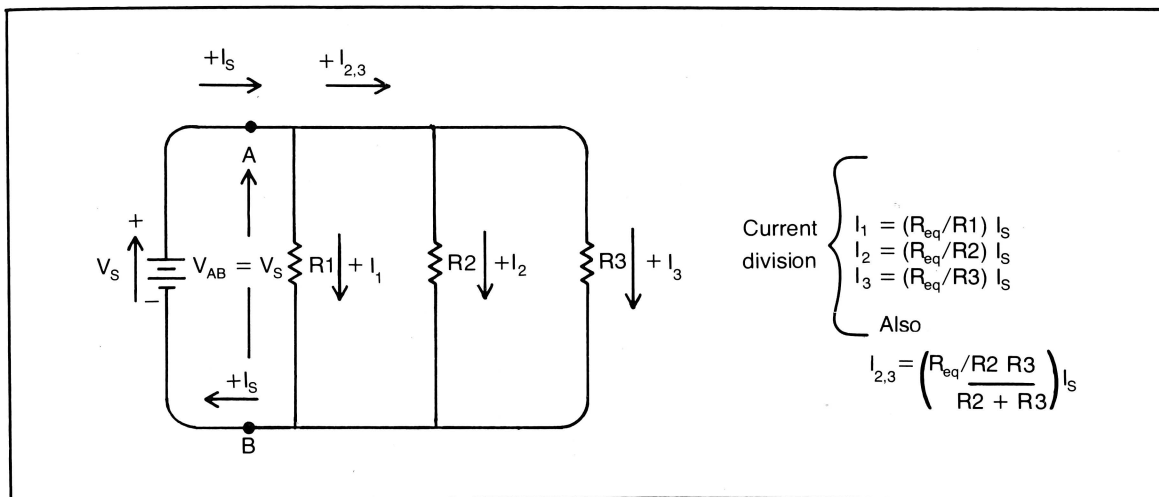


Fig. 5-7. Calculation of the current in each branch depends on the inverse ratio of branch resistance to equivalent resistance.

shunted through the smaller 10 ohm resistor. This shunting concept is important, as you will encounter it in practical everyday circuits. The shunting element (which may not be a resistor as such) in a sense “steals” current from the larger resistance. This principle is used in the transistor switching configurations that we will discuss in the next chapter.

Resistor Values, Ratings, and Uses

In most of your experimental work you will be using the common carbon resistor. These inexpensive components have a number of ratings:

- ☐ The resistance value of the resistor, expressed in ohms.
- ☐ The tolerance or variation of that value, expressed as a \pm percent.
- ☐ The power rating of the resistor, expressed in watts or fractions thereof.

Resistors also come in a variety of other forms: precision resistors for test equipment, wire-wound power resistors for high wattage applications, resistor networks for specialized or custom applications, and even resistor DIP packages which are used in bus-oriented (multiple, parallel line) computer circuits. While our concern is with

carbon resistors, what follows applies to these other types of resistors as well.

Figure 5-8 illustrates a typical carbon resistor, on which there are several color-coded band markings. The first three bands indicate the *resistor value*, and the fourth its *percent tolerance*. The diameter of the resistor and its overall size indicates its *power rating*.

Table 5-1 shows the color codes used to determine resistor values and tolerances. Referring to the table and to Fig. 5-8A and B, you'll note that the first two color bands represent the *significant digits* of the resistor value, and the third band gives the *multiplier*.

For instance, red-violet-orange would signify 27 times 10^3 or 27,000 ohms. For brevity, this is written as 27 K. The silver fourth band means that this value may vary 10% above or below the nominal 27 K value. The absence of a tolerance band indicates a 20% tolerance. A gold tolerance band means 5% variation in value.

At one time you had to pay a premium for 5% resistors, but no longer. In fact, most of the carbon resistors sold now, even those from surplus and mail-order sources, are 5% tolerance components.

You'll find that the 5% and 10% tolerance ratings are more than adequate for most work. In fact, there are some circuit applications, such as in cur-

Table 5-1. Color Codes for Calculating Resistor Values and Tolerances.

Color	1st Band	2nd Band	3rd Band (multiplier if used)	4th Band % Tolerance
Black	0	0	1	
Brown	1	1	10	
Red	2	2	100	
Orange	3	3	1,000	
Yellow	4	4	10,000	
Green	5	5	100,000	
Blue	6	6	1,000,000	
Violet	7	7	---	
Gray	8	8	---	
White	9	9	---	
Gold	-	-	0.1	5%
Silver	-	-	0.01	10%

rent limiting and transistor saturation, where resistors may vary as much as 40 or 50% from their recommended value and still do the job satisfactorily.

Resistor Power Ratings: In order to understand power ratings, we have to digress a bit to define power itself. In a mechanical system, power is defined in terms of applying a force against a

weight to move it a distance in a certain unit of time. In everyday terms, it takes more power to move a weight rapidly against the force of gravity than it does to do so slowly.

In an electrical system, power is defined in terms of applying a motive force, voltage, on a quantity of charge so that a certain amount of charge flows per unit of time. This translates into applied

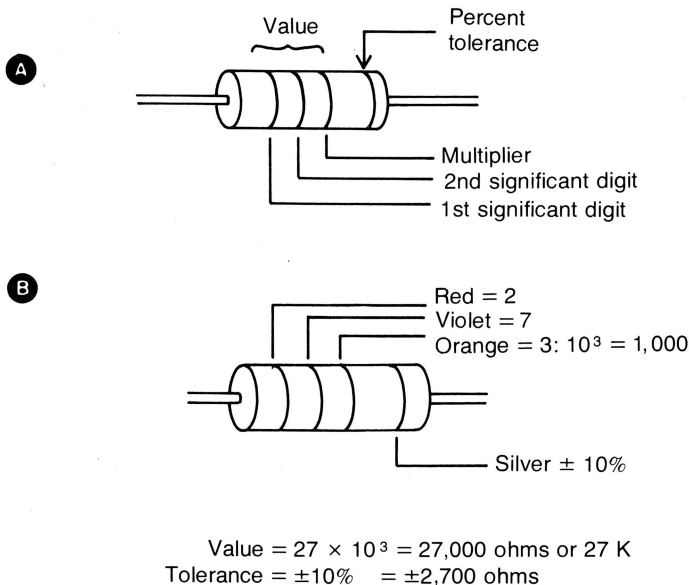


Fig. 5-8. Typical four band resistor in (A), with an example for calculating value and tolerance in (B).

voltage times resulting current flow. This makes physical sense if you remember that you are doing work (applying voltage) on a quantity of charge in order to move it through an electrical resistance. The basic relationship is:

$$\text{Power} = \text{Voltage} \times \text{Current}$$

Let P stand for power. We write this relationship, and then apply Ohm's Law to rephrase it:

$$P = VI \quad 5-9a$$

$$P = V \times V/R = V^2/R \quad 5-9b$$

$$P = I_R \times I = I^2R \quad 5-9c$$

These equations are equivalent by Ohm's Law. They all express the power needed to move current through a resistive element under a specified voltage. The unit of electrical power is the *watt*. It is defined as the amount of power used to cause one ampere of current to flow under the influence of one volt. Note that in Equation 5-9b and Equation 5-9c that power consumption of a resistor does not go linearly but as the square of voltage or current. For example, given a fixed resistance, it takes four times the power to cause a doubling of current flow through it.

As far as power ratings go, $\frac{1}{4}$ to $\frac{1}{2}$ watt ratings are fine for most applications. ($\frac{1}{2}$ watt resistors are typically $\frac{1}{8}$ of an inch in diameter and $\frac{3}{8}$ of an inch long.) Rarely will you need one watt resistors in digital circuits, although they are common in such applications as power amplifiers. For printed circuit work where low wattage prevails, $\frac{1}{4}$ or even $\frac{1}{8}$ watt ratings are preferred because of size considerations. One-half watt resistors are a better choice for breadboard work, though. This is not because you need the higher power rating, but because the leads are often thicker, making these resistors more rugged for repeated use.

The power rating of a resistor has a very specific meaning. Resistors spend or consume electrical power, and this power has to go somewhere. Since resistors do not store energy, they must dissipate it. They do so by radiating the electrical power in another form: *heat*. Often the heat

generated can be put to a useful end, such as in electric stoves, coffee makers, space heaters and similar appliances.

In electronic circuits, though, this heat is a potential problem rather than a useful end product. As a result of heat generated by the resistive elements in a circuit, ambient temperature about the circuit rises. The operation of certain components may be erratic under the thermal stress, causing overall faulty circuit function, or even outright failure. However, thermal problems can be overcome by proper circuit design (thermal compensation), passive radiative cooling (heatsinks), and forced convective cooling (fans).

Uses. Besides producing heat, useful or not, resistors perform numerous other circuit functions, a few of which will be mentioned. For our purposes, resistors will be used to limit current flow through other components such as diodes, transistors and digital ICs. Resistors are employed to *bias* transistors into saturation, so that they will operate properly as switches. They are also key components in determining the time constant of a circuit, which are useful for timing, clock signal generation and measurement. An example of measurement is the common joystick or game paddle.

Further, the voltage division principle can be applied to more sophisticated measurement such as analog to digital (A/D) conversion using resistor networks. Also, resistive transducers—components which change their internal resistance in response to changes in such quantities as light, temperature, and pressure—are the basis of laboratory measurement and industrial monitoring systems. A few experiments involving the construction and testing of simple resistive transducers will be presented in Chapter 11.

Examples of Resistive Circuits

To conclude this section, two examples of resistor circuits are given. These employ a battery as a constant voltage source. (Also known as direct current sources, or dc for short.) You will be asked to find the equivalent resistance of series and parallel resistor configurations, as well as voltage drops, current flows and power dissipation.

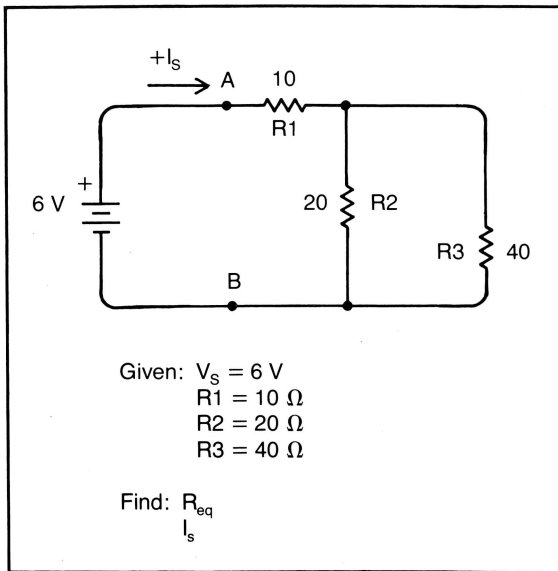


Fig. 5-9. Resistive circuit example.

The problem illustrated in Fig. 5-9 requires you to find the equivalent resistance and the source current, given the values of the three resistors and the source voltage.

The voltage source sees an equivalent resistance through points A and B, in Fig. 5-10A. In 5-10B the parallel equivalent resistance of R_2 and R_3 is:

$$R_{2,3} = R_2 \times R_3 / R_2 + R_3 = 800/60 = 13.3\text{ ohms.}$$

The total equivalent resistance and source current are:

$$R_{eq} = R_1 + R_{2,3} = 13.3 + 10 = 23\text{ ohms}$$

$$I_s = V_S / R_{eq} = 6/23 = .26\text{ amperes or }260\text{ mA}$$

The equivalent circuit is shown in Fig. 5-10C.

For the circuit in Fig. 5-11, find R_{eq} and I_s . Also, find the power dissipated in resistor R_2 , and the voltage drop across the set of parallel resistors R_4 , R_5 and R_6 ($R_{4,5,6}$).

Begin by redrawing the circuit, as in Fig. 5-12A. Again, you look in from the viewpoint of the voltage source, so that you see the same equivalent resistance that it sees. There are several ways of

solving this problem, with the most direct one shown here. First, find $R_{4,5,6}$ by direct inspection. It is equal to one-third the value of one of them, as mentioned before. The total series resistance in the right leg of the circuit, R_x , is illustrated in 5-12B. Finding R_x permits you to calculate the current in this leg, and therefore the power dissipated in R_2 and the voltage drop across $V_{4,5,6}$.

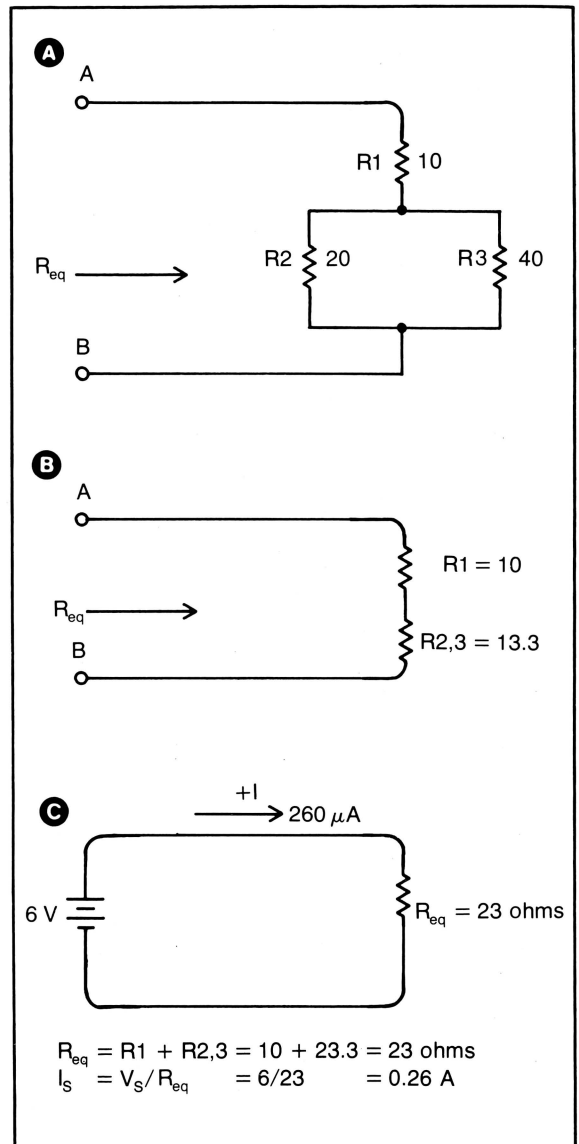
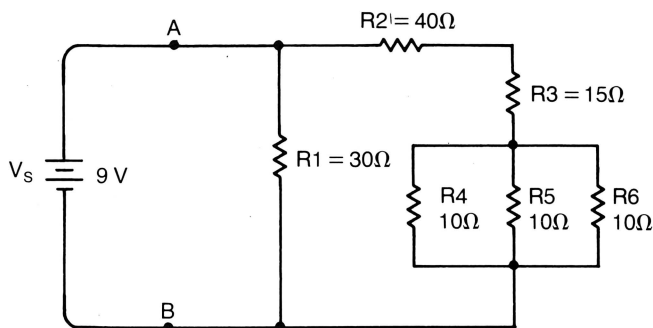


Fig. 5-10. Solution for Fig. 5-9.



Given: Resistor Values R1-R6 above
 $V_S = 9\text{ V}$

Find: R_{eq} and I_S
 Power in R2 (P_{R2})
 Voltage drop across R4,5,6 ($V_{4,5,6}$)

Fig. 5-11. Another resistive circuit example.

$$R_x = R2 + R3 + R_{4,5,6} = 40 + 15 + 3.3 = 58.3 \text{ ohms}$$

$$V_{4,5,6} = (R_{4,5,6}/R_x)V_S = (3.3/58.3)9 = .51 \text{ volt} \quad (510 \text{ mV})$$

$$V_{R2} = (R2/R_x)V_S = (40/58.3)9 = 6.17 \text{ volts}$$

$$P_{R2} = (V_{R2})^2 / R2 = (6.17)^2 / 40 = .95 \text{ watts}$$

R_x is R1 in parallel with R_x , so that

$$R_{eq} = (30)(58.3) / 88.3 = 19.8 \text{ ohms, and}$$

$$I_S = V_S / R_{eq} = 9/19.8 = .45 \text{ amp or } 450 \text{ mA}$$

The final equivalent circuit is shown in Fig. 5-12C.

In the second example, you could have calculated I_x , the current in the right leg, in order to find P_{R2} from the equation $P = I^2 R$. However, it generally seems easier to apply the voltage division rule than the current division method, perhaps because it is intuitively easier to take direct ratios for voltage division than inverse ratios for current division.

CAPACITANCE, TIME CONSTANTS AND REACTANCE

Capacitance, both in the form of an electronic component and as a property of all components, is

an essential concept. In this section, just as with the resistor, you will be introduced not only to the component itself, but to a number of related ideas.

Capacitors are used in electronic circuits, digital and analog, for many different purposes. Also, the behavior of other components—diodes, transistors and digital ICs—is influenced by their *internal* capacitance. Of particular importance is that by understanding capacitance, you'll have an intuitive grasp of associated concepts: the time constant and reactance.

A fair amount of space is dedicated, therefore, to describing capacitance, first by a hydraulic analogy, and later by simple RC (resistor-capacitor) circuits. Some example applications are mentioned, along with values and ratings of capacitors.

The Hydraulic Model

As the name suggests, a capacitor stores something, be it electrical charge or a tangible fluid like water. The idea of capacitance is presented in Figs. 5-13 and 5-14.

Consider a vessel into the top of which water flows from a pipe (pipe A) and out of the bottom of

which water exits from a second, smaller pipe (pipe B). See Fig. 5-13A. In this analogy, water represents charge and the size and shape of the vessel represent capacitance. The main thing to understand in this system is how the vessel (fluid capacitor) fills and empties (charges and discharges) given a certain flow of water from pipe A. The rates of filling and emptying are *exponential*.

That is, tank volume does not change in a straight-line fashion, but rather follows a gentle curve that approaches some equilibrium. The equilibrium state for this system is one in which the level of the water remains constant for a given flow from pipe A.

Let's see how the tank fills, first. Refer to Fig. 5-13B. The tank is empty and there is no water

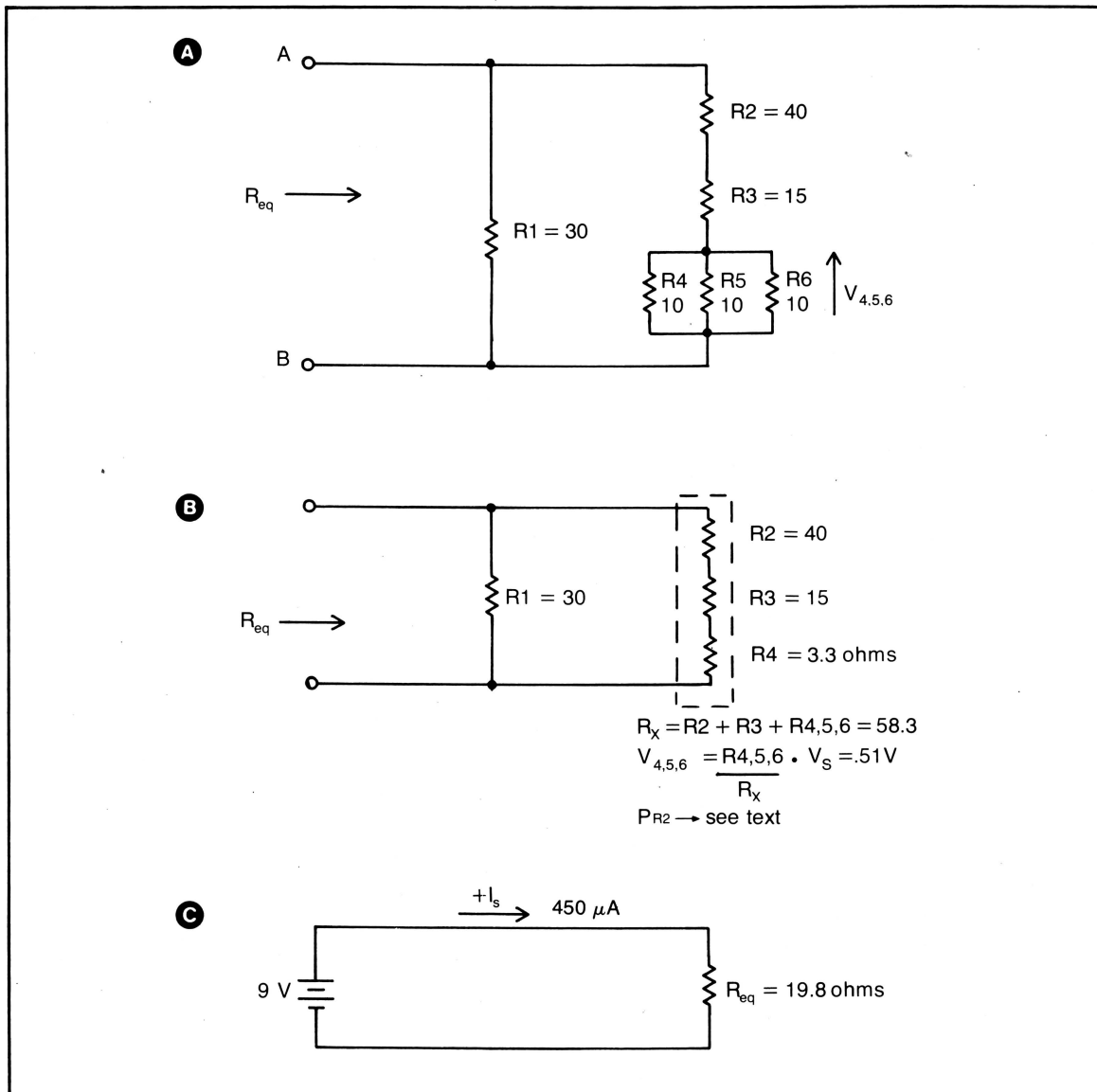


Fig. 5-12. Solution for Fig. 5-12.

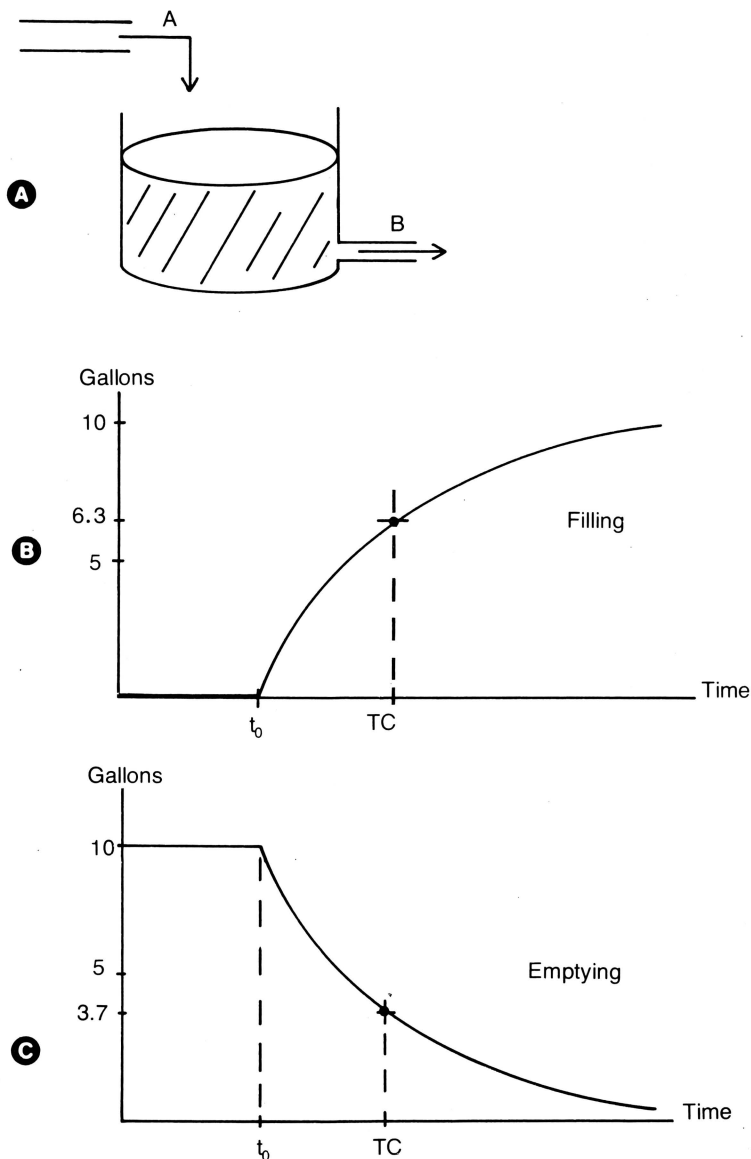


Fig. 5-13. Hydraulic model of capacitance. (A) Physical model. (B) Exponential filling curve. (C) Exponential emptying or discharge curve.

coming in from pipe A. Then we turn on pipe A at time t_0 . Assume that there is a steady flow of water into the tank from pipe A, and that the initial outflow of water into the tank from pipe B is low. Even though water exits from B, the tank will accumulate water via the greater inflow from A. The height (and

volume) of water in the tank at first rises fairly rapidly. But as the height of the water increases it will exert an increasing downward pressure at the outlet, and hence the outflow from pipe B will steadily increase. Eventually, the inflow from A is matched by the outflow from B. They are equal, and

the volume in the tank is constant, in an equilibrium or steady state.

As you see in Fig. 5-13B, there is a sharp rise in tank volume at first, and then a gradual fall-off in the rate of filling as it approaches the steady state volume. Indicated in the graph is a characteristic time for this system, represented by TC, the time constant. This is the time it takes to fill to about 63% of the equilibrium volume, starting from when pipe A was first turned on, t_0 . More will be said of the time constant shortly.

Let us say that this equilibrium volume (V_{eq}) is 10 gallons, and we leave the system alone for a

while in this condition. The tank is fully *charged* at this rate of inflow, and the water volume will not change if we keep pipe A inflow constant.

Then suddenly, we turn off the inflow from pipe A. Refer to Fig. 5-13C. We'll call this turn-off time t_0 also. Immediately, the tank starts to empty through pipe B. At first, the outflow is rapid due to the height and weight of the water column. The level falls quickly at first, as you see from the steep slope of the curve in the figure. As the level falls, so does the height and weight of the water column. Consequently, outflow through pipe B declines steadily and the rate of emptying likewise slows.

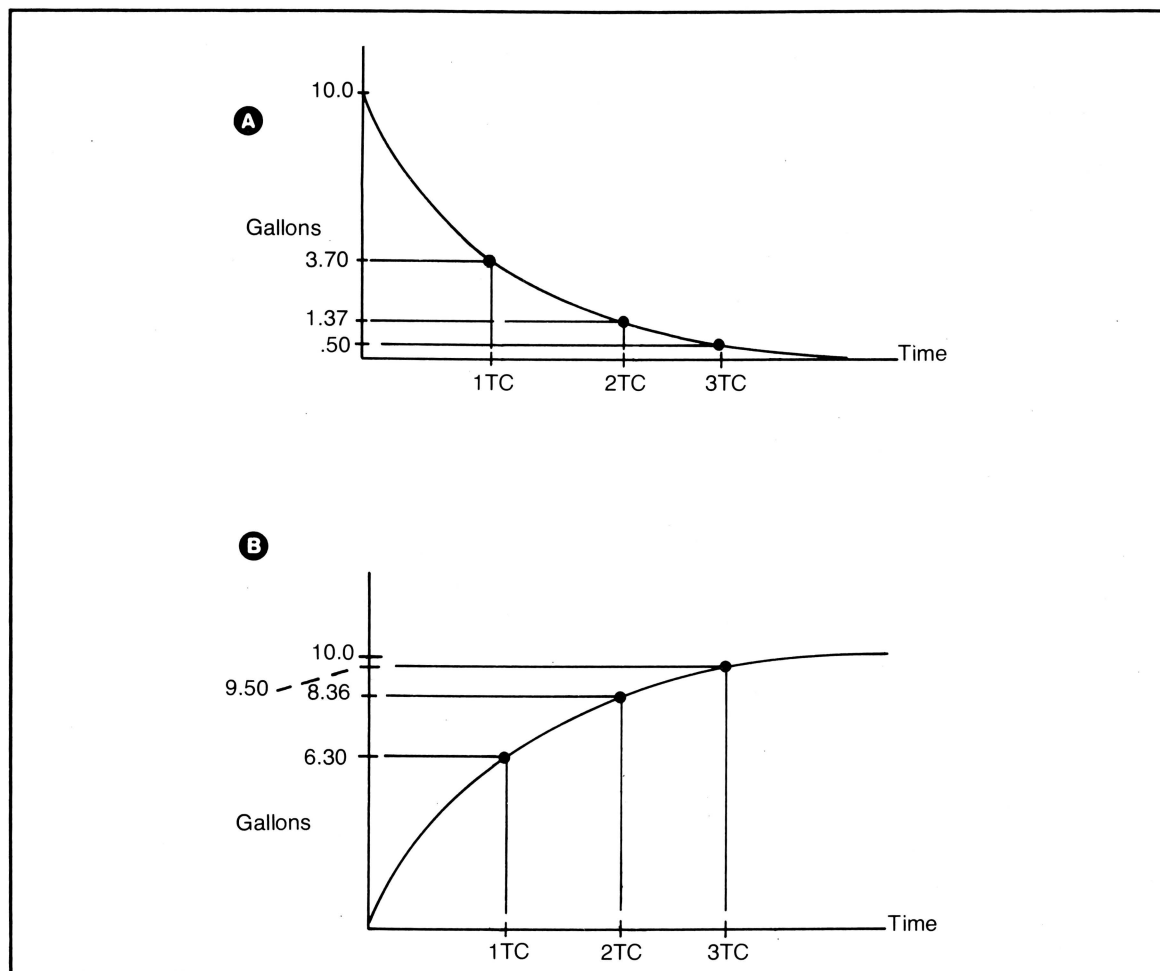


Fig. 5-14. (A) Redrawn discharge curve for the tank, with volume remaining indicated various times, expressed as multiples of the time constant. (B) Redrawn charging or filling curve for the tank, with volumes at TC multiples indicated.

This falling off of the rate of emptying is reflected in the decreasing slope of this *discharge* curve. This situation is similar to tank filling; the same principles operate in the reverse direction. And like the filling curve, the emptying curve in Fig. 5-13C also has a characteristic time value or constant, TC, measured from time zero when the inflow pipe was turned off. This is the time it takes for the tank to discharge to 37% of its equilibrium volume (V_{eq}) of 10 gallons.

These time constants for filling and emptying are equal because the time constant is a fundamental property of such systems as this. It takes one TC for the tank to fill to 63% of its V_{eq} . Or, you could say that it takes one TC for it to fill to within 37% of this volume. When the tank is discharging, it takes one TC for it to fall to 37% of its V_{eq} , or alternatively, to be 63% discharged.

In other words, most of the change in tank volume, whether charging or discharging, takes place during one time constant.

To more fully describe such systems, we have to use some math. The hydraulic system described, and similar systems such as charging and discharging electrical capacitors, are known as *first-order* systems. The charging and discharging of capacitors in these systems is governed by a time constant and is related to time in an exponential fashion. For the emptying tank, the volume in the tank will decrease according to Equation 5-10a:

$$V = V_0 e^{-t/TC} \quad \text{5-10a}$$

where V is tank volume at any instant in time.

V_0 is the current tank volume.

e is the *natural base*, which equals 2.71828.

t is the time interval from when V_0 was measured to when V is measured.

TC is the system time constant.

This equation describes the steady decline or *decay* of tank volume towards zero with the passage of time. The natural base, e , is so called because it is used in many physical systems to describe time varying quantities. Base 10 could have been used, but this would have added another numerical factor

to the exponent. The exponent of e , t/TC , is negative. Remember that any number raised to a negative power is really the reciprocal: $10^{-2} = 1/10^2 = 1/100 = .01$, for instance. Since the exponent increases in time, the value $e^{-t/TC}$ becomes smaller. Therefore, V becomes smaller with the passage of time.

The shape of this Volume versus Time curve is shown again in Fig. 5-14A, which is a slightly embellished version of the emptying curve of 5-13C.

This exponential decay curve and the equation that describes it can be understood if we consider the passage of time in *multiples* of the time constant. (The actual value of TC, be it minutes, seconds or hours, is unimportant.) That is, let's measure time intervals in integer multiples of TC: $t=n(TC)$, where n is a whole number, 1, 2, 3, etc. Then the ratio t/TC would be just 1, 2, 3, for each such interval. At $t=0$, t/TC would be 0, and the exponential term would be:

$$e^{-0/TC} = e^{-0} = 1$$

At time $t=1TC$,

$$e^{-1TC/TC} = e^{-1} = 1/e^1 = 1/2.718 = .368$$

The equation tells you that after one TC, the volume will decrease to about 37% of its original value. If we started at 10 gallons, there would be 3.7 gallons by time $t=1TC$, as shown in Fig. 5-14A.

What happens as more time passes? Again using multiple of TC, we can express Equation 5-10a more conveniently as:

$$V = V_0 (.368)^n \quad \text{5-10b}$$

where n =time in multiples of TC.

From the graph in Fig. 5-14A you can see that by 3TCs the tank is almost empty. If you refer to Table 5-2, third column, you'll note that after 3TCs the actual decline is down to 5% of the starting volume, and after five TCs down to less than one percent.

One more thing about the discharge curve. You could start anywhere on the curve, at some arbitrary

Table 5-2. Table of the First Order System Discharge and Charging Factors. (If Time is Measured in Integer Multiples of TC, the Values for $t=0$ to $t=5TC$ are as Listed. A First Order System is Practically in Equilibrium by 4 or 5 TCs.)

$t=n(TC)$	t/TC	$e^{-t/TC}$ decay factor	$(1 - e^{-t/TC})$ charging factor
0TC	0	1.000	0.00
1TC	1	.368	.632
2TC	2	.140	.860
3TC	3	.050	.950
4TC	4	.018	.982
5TC	5	.007	.993

rary volume of say 7.5 gallons. Again, it would take an interval of 3TCs for the volume to decrease to 5% of this volume, namely $(.368)^3 \times 7.5$ or about .38 gallons.

That is, whatever volume we choose to start with (V_0), we know that there will be $(.368)^n$ of that volume remaining after n time constants. It is not the starting volume, but the time constant that matters in determining the amount of change in the system with time.

In filling the tank, on the other hand, we use a modified form of Equation 5-10a:

$$V = V_{eq} (1 - e^{-t/TC}) \quad 5-11a$$

The factor $(1 - e^{-t/TC})$ applies to a charging or filling system. As with the discharging equation, we can rewrite the exponential term using multiples of TC for time. Just substitute directly for the natural base term:

$$V = V_{eq} (1 - (.368)^n) \quad 5-11b$$

The charging factor starts out at 0 at time zero ($t = 0TC$). As time passes it approaches unity because $e^{-t/TC}$ get smaller and smaller. The equation tells you, then, that V gets closer to the equilibrium volume V_{eq} with time. This is expressed in the graph of Fig. 5-14B, and by the actual numerical values in the fourth column of Table 5-2. (To get these values just subtract the values in the third column from one.) As you'd expect, the percentage changes in volume are comparable to the discharging case. Starting from zero volume, the

volume of a filling tank is about 63% of the final equilibrium volume (or to within 37% of it) after 1TC. The tank is 95% filled by 3TCs, and so on.

The factors that influence that *time constant* will be now mentioned. In the hydraulic model the capacitance of the tank—its *fluid inertia* or tendency to fill and empty quickly—depends upon its geometry. A tall thin tank would have a smaller capacitance than a low squat one. The skinny tank would reach its equilibrium volume for a given inflow rate fairly soon, and this volume would be relatively low. This is because the tall column of water would more quickly exert a higher pressure on the outflow pipe.

Besides the capacitance of the tank, there is also the resistance of the outflow pipe. For a given fluid capacitance, a system with a narrow outflow pipe will have a longer TC than one with a large diameter pipe. The system with the thin pipe will take a longer time to reach the equilibrium volume, and this volume will be larger.

The extremes of these two situations are illustrated in Fig. 5-15A and B. In the case of a skinny tank/wide pipe system, the time constant will be low because equilibrium will be reached sooner. A squat tank/thin pipe system will have a much longer TC. We can express the time constant as a product of tank capacitance times pipe resistance, given in Fig. 5-15C. Generally,

$$\text{Time Constant} = \text{capacitance} \times \text{resistance}$$

This discussion of a hydraulic capacitor (tank)/fluid resistor system was meant to give you a physical understanding of the concepts involved, most important of which is the time constant. To some extent, this physical analogy is directly applicable to the electrical capacitor. As you will appreciate, the rate of inflow is analogous to charging current flowing under the influence of an applied voltage. The equilibrium volume (more accurately, the weight of the water column) is analogous to the time-dependent voltage across an electrical capacitor. Further, the electrical RC time constant expressed in seconds is in fact equal to resistance in ohms times the capacitance in farads (the unit of

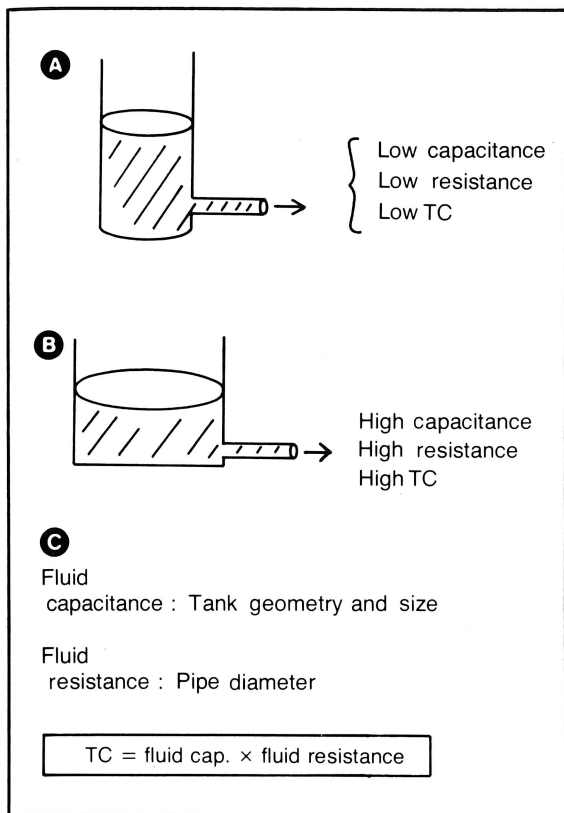


Fig. 5-15. (A) Tall thin tank with low resistance outlet—low TC. (B) Low squat tank with high resistance outlet—high TC. (C) The time constant for such fluid systems is expressed in terms of an RC product (resistance \times capacitance).

electrical capacitance). These concepts will be discussed and expanded in the following section.

Electrical Capacitance and the RC Time Constant

Most of the things you should know about electrical capacitors are implied by the concept of the time constant, already introduced, and by the *capacitor component law*. The component law is given in Equation 5-12:

$$Q = CV \quad 5-12$$

where Q is the accumulated charge in coulombs.

C is the capacitance in farads, F.

V is the applied voltage across the capacitor.

This equation states that a one farad capacitor will store one coulomb of charge when one volt is applied. One farad is a large unit of capacitance, but is used for simplicity in our examples. The equation could also be read as stating that a one farad capacitor with one coulomb of stored charge has a voltage potential of one volt across it at equilibrium: $V_{\text{cap}} = Q/C$.

Referring to Fig. 5-16A, you can see that the capacitor itself is represented by two metal plates separated by a thin layer of nonconducting material, a *dielectric*. In a large capacitor this plate and foil sandwich is wrapped up into a compact cylindrical “jelly roll” in order to save on space. The larger the plate area, the more charge that can be stored. Also, the thinner the dielectric, the greater the attraction of stored charges on either plate. Both these factors, surface area and spacing, as well as the nature of the dielectric itself, influence capacitance. Large plates spaced close together constitute a relatively larger capacitor than smaller plates further apart.

The property of storing charge is what defines capacitance, as suggested by the equation. Any time two conducting surfaces, however small or irregular, are separated by a space, the potential for storing charge exists. Therefore, not only capacitors, but virtually all electronic components have some innate capacitance. Even two parallel wires placed side by side have a small but measurable capacitance between them!

The physics of capacitance is complex, but you can understand just what a capacitor does from Equation 5-12, from what you learned in the hydraulic model, and from the simple circuit in Fig. 5-16.

In Fig. 5-16A, there is no charge on the capacitor plates. When the battery leads are connected to the capacitor, current (+I) flows through the circuit and charges accumulate on the plates. The reason for this current flow is that plus charges from the positive terminal are attracted to the minus charges from the negative terminal. (Remember, it is valid to talk of a virtual flow of positive and negative currents going in opposite directions.) The accumulation results in a rush of plus

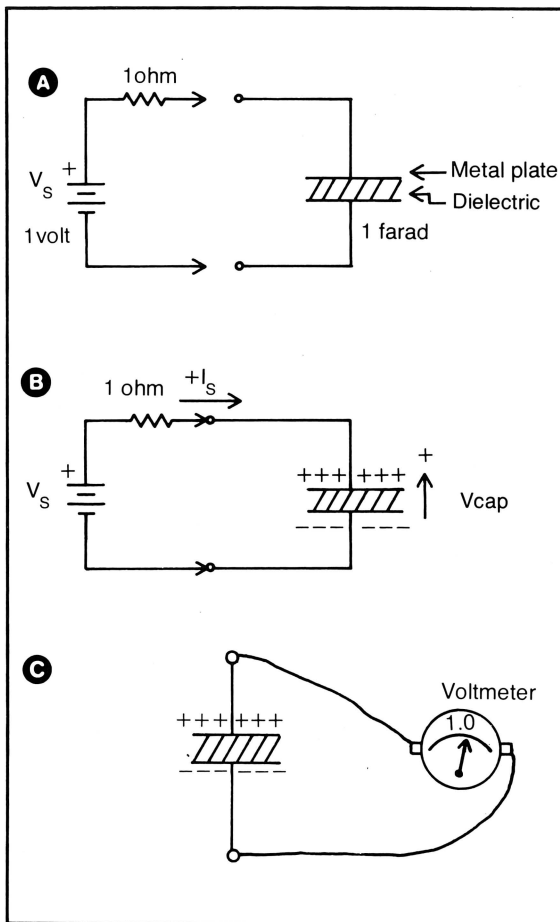


Fig. 5-16. (A) Uncharged, (B) charging, and (C) fully charged capacitor at equilibrium voltage. See text for discussion.

and minus charges from the respective battery terminals towards the upper and lower plates, as in Fig. 5-16B. So even though the dielectric will not conduct current because it is an insulator, current does flow due to this attraction of unlike charges across it. It is just as if a current were passing through the capacitor while it is charging.

This *charging current* is high initially, but decreases with time. As plus charges accumulate on the top plate, they begin to repel other plus charges coming from the positive terminal; likewise for the minus charges accumulating on the bottom plate. Eventually, an equilibrium is reached in which the attraction of unlike charges across the dielectric is

balanced by the repulsion of accumulated charges on the plates.

In this steady state, no further current flows. We can state this another way. From Equation 5-12 the acquired capacitor voltage potential is now equal to the applied voltage in this equilibrium state. From the standpoint of the resistor there are two 1-volt voltage sources (the battery and the charged capacitor), the positive terminals of which are attached to either end. Therefore, the potential difference across the resistor at this point is zero, and again no current flows.

At this equilibrium state, you could remove the ideal capacitor from the circuit and it would maintain the stored charge indefinitely. Real capacitors have a small *leak* of current across their plates because the dielectric is not a perfect insulator. Still, your ordinary surplus capacitor will maintain its charge for a time, and you could measure a voltage across it just as in Fig. 5-16C. In the ideal case the meter would read 1-volt.

Given the circuit of Fig. 5-16A, the next question is, "Just how long does it take for the charging current to fully charge the capacitor to one volt?" That depends on the values of the resistor and the capacitor, as you may already have guessed.

For instance, if the capacitance were high, more accumulated charge would be necessary to create a one volt potential across the capacitor ($Q=CV$). If the resistance also increased, this would reduce current flow and charging time would correspondingly increase. The time it takes to reach equilibrium, then, is directly proportional to the values of the resistor and the capacitor. Obviously, we're leading up to the circuit time constant. It is expressed by the same form of equation introduced at the end of the last section. For this circuit, the RC time constant is:

$$TC = R \times C \quad 5-13$$

where TC is the time constant in seconds.

R is the resistance in ohms.

C is the capacitance in farads.

For our battery charged capacitor, the time

constant is $1 \text{ ohm} \times 1 \text{ farad}$, or 1 second. After 3TCs it is 95% charged (.95 V). The voltage versus time curve in Fig. 5-17 has the exponential shape of a first-order system. The equation for the voltage that you would measure at any time during the charging process is given by another familiar equation:

$$V = V_s(1 - e^{-t/RC}) \quad 5-14$$

where V is 1 volt and the TC ($=RC$) is 1 second.

Just as important as the voltage curve, is the

curve for charging current. Figure 5-17B shows how this current decreases according to the exponential decay relationship:

$$I_s \text{ or } I_{\text{cap}} = I_0 e^{-t/RC} \quad 5-15$$

Note that I_{source} is the same as the current going through the capacitor, as there is only one loop in the circuit. I_0 is the current flowing in the circuit at time zero. How is it determined? By realizing that at t_0 there is no charge in the capacitor so that $V_{\text{cap}} = 0$. Those were our starting conditions. Therefore, the full voltage of one volt is present across

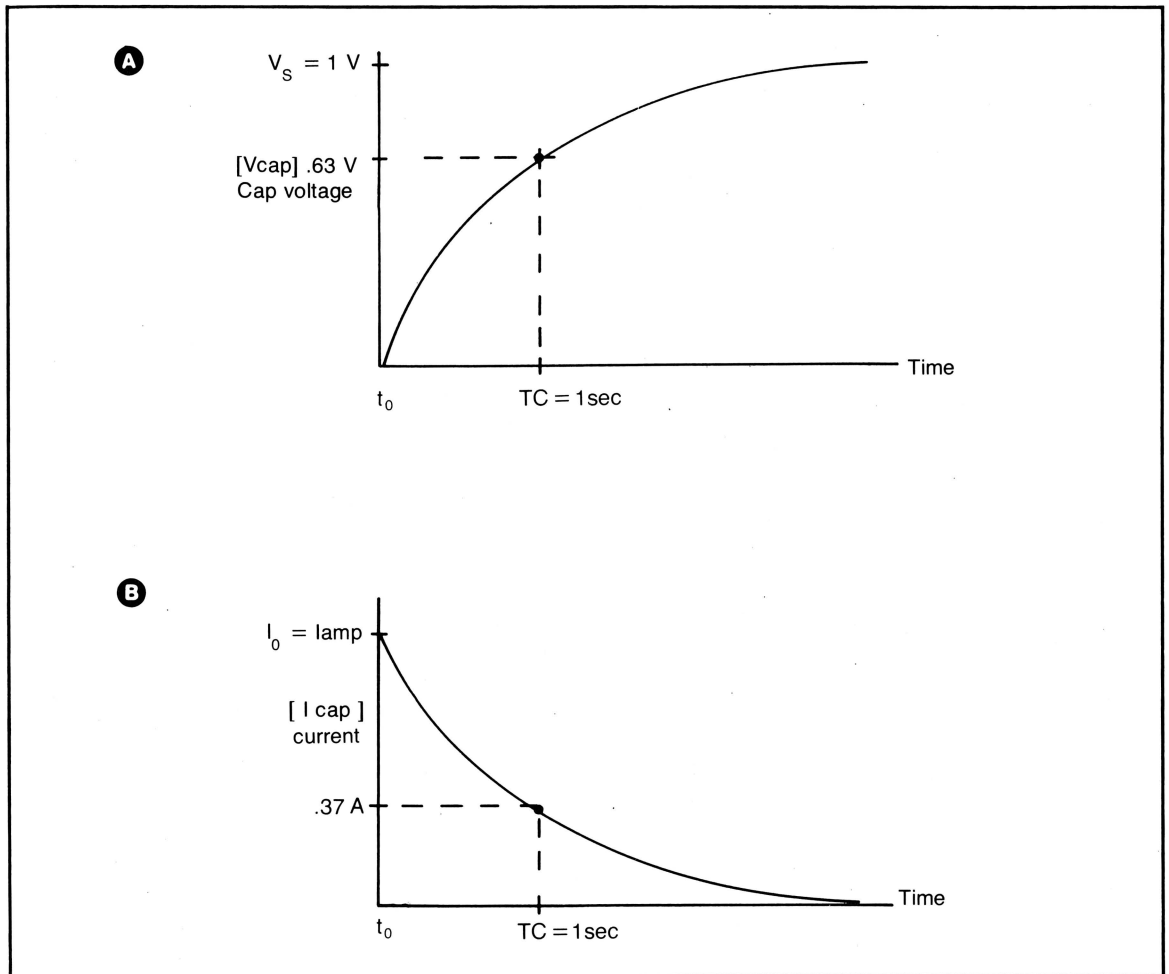


Fig. 5-17. During the charging sequence in Fig. 5-15, the capacitor voltage and current have the curves shown in (A) and (B) respectively.

the resistor at the instant of t_0 . This one volt differential causes a one amp current across the one ohm resistor at this time, but thereafter declines exponentially to zero. Again, current declines as V_{cap} increases towards the equilibrium or charging voltage, because of the decreasing potential difference across the resistor, as just explained.

To examine the discharge curve, we will connect the fully charged capacitor of Fig. 5-16C to

ground through another one ohm resistor, shown in Fig. 5-18A. Ground means some type of *sink* that can absorb charge and dissipate it. Earth ground is an example, though a metal chassis or other extended metal surface may be thought of as a ground. We could also connect the leads by a wire, as indicated by the dashed line, and get the same result. When the capacitor discharges through the resistor, current flows as the plus and minus charges on

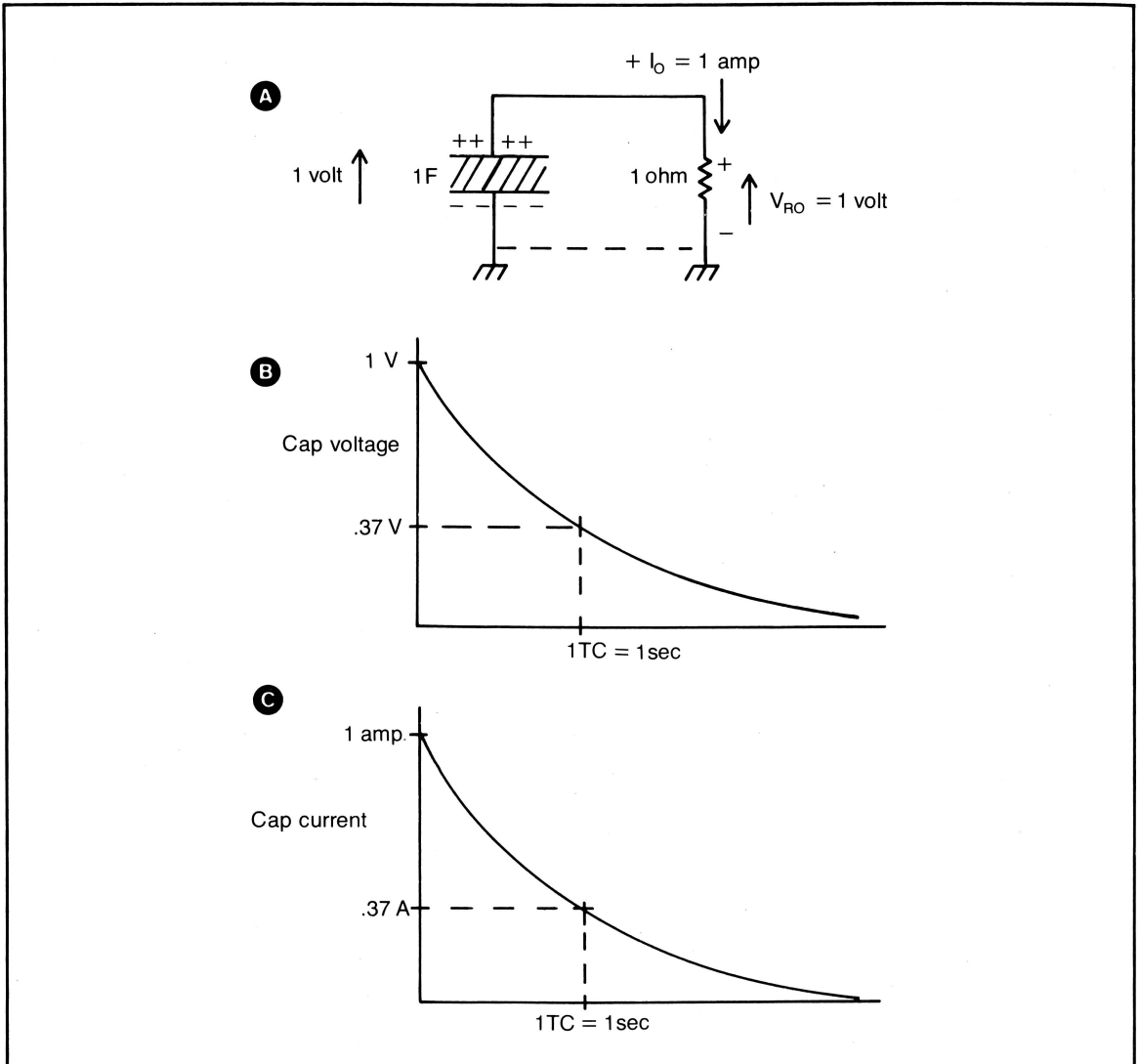


Fig. 5-18. For the discharging capacitor in (A), the exponential decline of voltage and current will look like (B) and (C), respectively.

the plates rush towards one another. Stored charge, which is responsible for the capacitor voltage, decreases. As a result, V_{cap} declines from one volt and eventually reaches zero volts. The fall-off in current and V_{cap} is given by:

$$I = I_0 e^{-t/TC} \quad 5-15$$

$$V_{cap} = V_0 e^{-t/TC} \quad 5-16$$

V_0 is the voltage at time zero, which is the 1 volt equilibrium voltage we started with from Fig. 5-16C. The graphs for V and I against time are plotted in Fig. 5-18B and C. Both V and I are effectively zero (within 1%) after 5TCs, or about 5 seconds.

Reactance

So far we have talked about constant voltage or dc sources. What about ac sources? How does capacitor current vary when the applied voltage varies? The answer to this is given by the component law, or more specifically its *time derivative*:

$$\begin{aligned} Q &= CV \\ dQ/dt &= C dV/dt \\ \text{or} \\ I &= C dV/dt \end{aligned} \quad 5-17$$

When discussing the charging and discharging of a capacitor it is usually more meaningful to speak about the current flowing through it than about the number of coulombs it stores. Equation 5-17 allows us to do just that, and it provides a few other insights as well. This version of the component law states that the effective current flow through a capacitance will be greater if,

□ dV/dt , the rate of change of applied voltage, its *frequency*, is higher.

□ C , the value of the capacitor, is higher.

The physical sense of this is as follows. Assume some capacitor is in equilibrium with some fixed voltage applied. This voltage then suddenly changes. This quick change in voltage will cause an equally rapid change in the accumulated charges to

come to equilibrium, and a current will result, as just described in the dc model above. Slowly changing voltages will affect a correspondingly more gradual re-equilibration of charge, and less current will flow. In short, for a given capacitor value and voltage amplitude, a capacitor will pass more current with high frequency signals than with low frequency signals.

Second, larger capacitors will allow more current flow than small ones. The reason for this is that large value capacitors are more capable of storing or delivering greater quantities of charge for any given level of voltage. This follows from Equation 5-12. Therefore, given a certain swing in the polarity or magnitude of the voltage, large capacitors can provide greater swings in the accumulation or delivery of charges under this varying voltage, than can a small capacitor. This translates into larger swings in current. In short, for a given frequency and amplitude of applied ac voltage, large capacitors will pass more ac current than small capacitors.

According to the component law in its time derivative form, Equation 5-17, capacitors can be thought of as presenting a type of resistance to ac current flow, technically known as *reactance*. Capacitive reactance is less at higher frequencies, and is less for larger capacitors, because more current flows in either instance. Capacitive reactance, the resistive property of a capacitor that limits ac current, is given by:

$$X_C = 1 / 6.28 f \times C \quad 5-18$$

where X_C is capacitive reactance in ohms.

f is frequency in hertz (Hz or cycles per second).

C is the capacitance in farads.

For instance, at a frequency of 100,000 Hz (100 kHz) a 1 microfarad (μF) capacitor would have an effective resistance of $1/6.28(10^5 \text{ Hz})$ ($10^{-6} F$) or about 1.6 ohms.

A special case of high frequency signals is the square wave, more specifically the leading and trailing edges of the square wave. A one volt square

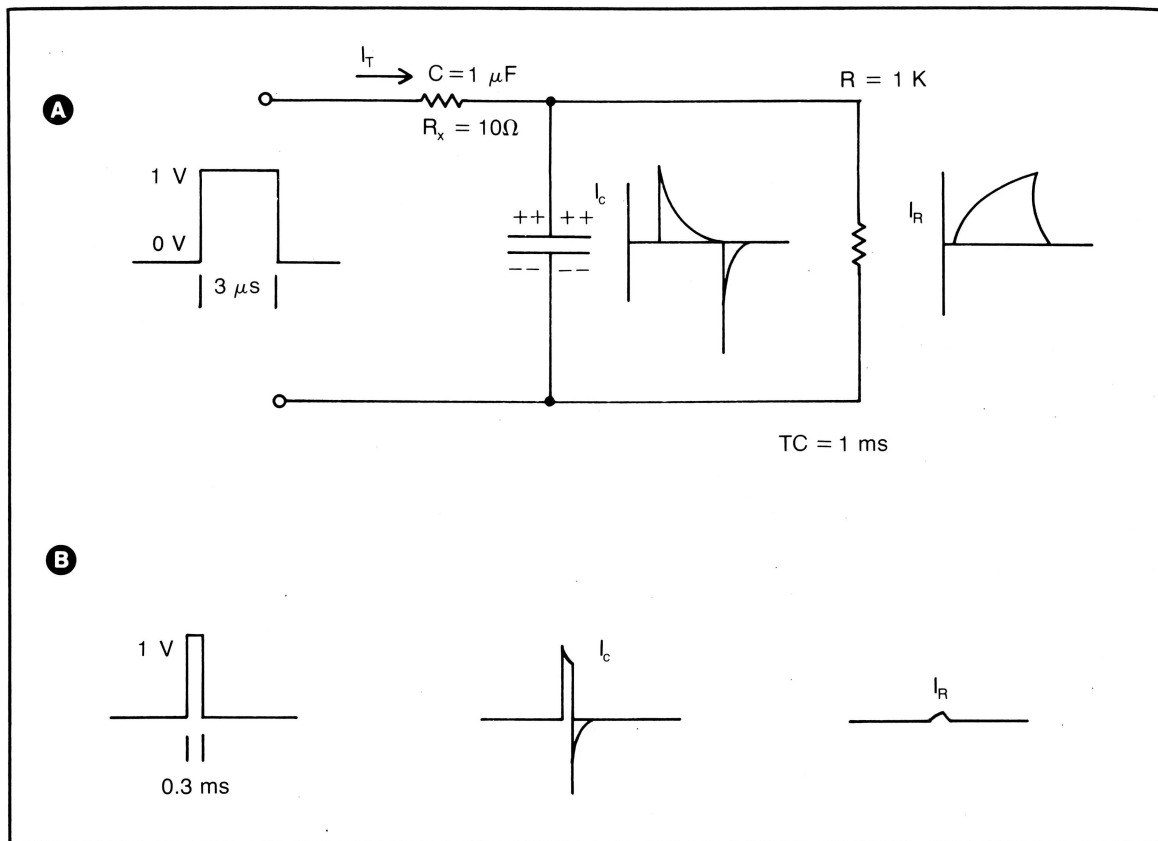


Fig. 5-19. (A) The shape of the current curves in the capacitor and resistor branches of a circuit in response to a single square wave. Here, the duration of the square wave is three times the TC. R_x is small enough to be ignored in calculating TC. (B) A very short pulse of duration much less than TC will result in negligible^x resistor current, I_R . See text.

wave signal, which swings sharply from 0 to 1 volt and back again, is shown in Fig. 5-19.

The hypothetical square wave will swing between its high and low values instantaneously— dV/dt would then be infinity! In reality, square waves have finite but very short rise and fall times, dV/dt for these edges is therefore still quite high. These edges are referred to as high frequency *unit pulses*, with a frequency often in the gigahertz range. The reactance, X_C , of even fairly small values of capacitance at the extremely high frequencies of these unit pulses is very low—a few tenths of an ohm or less. Therefore, a capacitor is essentially a short circuit at these high frequencies and is seen by the voltage source as a simple length of wire!

Figure 5-19 summarizes some of the properties of capacitive reactance that we've been talking about. The time constant for the circuit in Fig. 5-19A is $TC = 10\text{ ohms} \times 10^{-6}\text{ farads} = .001\text{ sec}$ or 1 millisecond (ms). The signal is a single square wave with an amplitude of one volt. The leading or rising edge of the signal is the high frequency unit pulse mentioned above. The capacitor is a reactance or *ac resistor* under these conditions. Virtually all the current flowing during the instant of this 0 to 1 volt transition appears through the capacitor. This is because the total current in the circuit, I_T , will be divided with most of it going through the effective short circuit presented by the capacitor. (Remember Kirchhoff's Current Law and current division.) As time passes, current decays in the

capacitor branch of the circuit (charge accumulation on the capacitor), and more current flows in the resistor branch, I_r .

At the end of the 3 ms square wave, the capacitor is nearly charged (95%). When the trailing edge occurs, 1 to 0 volts, the accumulated charge flows out of the capacitor to the ground or zero volt level presented by the signal source. Current through the capacitor branch now flows in the opposite direction as shown by the negative spike on the graph of I_c . (Charges are coming off the capacitor the same way they came in, but in reverse.) The *discharge time constant* is short because the internal resistance of the signal source of the square wave is very small. Therefore, most of the discharge current flows to this signal ground rather than through the resistor, and the discharge spike is very short.

If a very narrow square wave is used, say 1/10 the width of the one above, we can see the effect of reactance even more dramatically. As shown in Fig. 5-19B, for the same values of R and C the current wave form I_c appears as an up and down or biphasic spike. The positive part is simply the fore-

shortened charging curve. The negative discharge spike occurs again, but it is of lower amplitude because fewer charges accumulated during the short duration of the square wave. More interesting is the appearance of I_r . It barely rises above zero, before the input signal returns to zero.

If you understood in a qualitative sense everything that went on in Fig. 5-19, then you have a good grasp of the concepts presented. By the way, the reason for the 10-ohm resistor R_x is to limit the total current in the circuit (I_r) to around 100 mA.

Capacitor Ratings, Values and Uses

Capacitors come in several forms. The small value *disk* capacitor is shown in Fig. 5-20A. It has a value range from small fractions of a microfarad (10^{-6} F or μ F) to about 1 μ F or so. The maximum voltage rating is anywhere from 10 volts to around 100 volts. Larger applied voltage fluctuations would destroy the capacitor, either by creating a short or a permanent open circuit. Both value and voltage rating are usually printed on the disk. It is safe to assume that the tolerance is on the order of

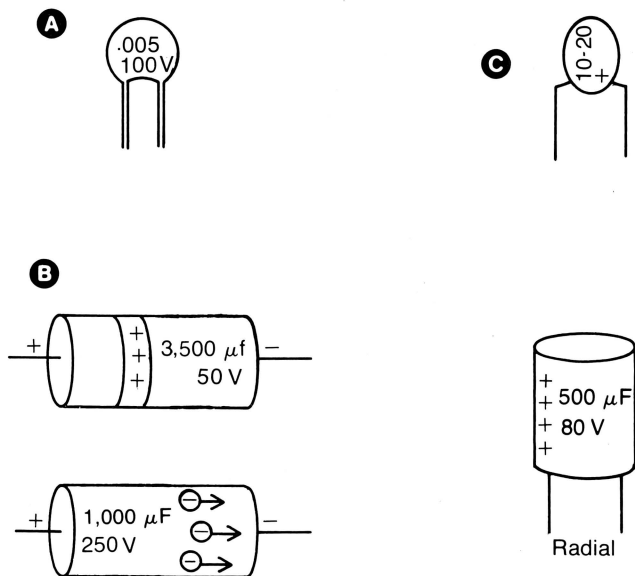


Fig. 5-20. (A) Disk, (B) electrolytic "can," and (C) tantalum capacitors.

10-20% for disk capacitors, unless otherwise specified on the component.

Electrolytic capacitors have a special dielectric which greatly augments capacitance for a given plate area and spacing. This type of capacitor is used in power supplies, amplifier output stages and other high power applications. Electrolytics are *polarized*. This means they can only function under conditions where the voltage does not reverse its polarity. The polarity of an electrolytic, or *can* as it is sometimes called, is marked on the component as shown in Fig. 5-20B. Typically, values may run to several thousands of microfarads, and ratings from a few volts to hundreds of volts. Both axial and radial forms exist, as indicated.

Special *computer grade* capacitors also exist, a common type being the tantalum capacitor. These can be recognized as little globular components comparable in size to small disks. Tantalums have high values for their size, and higher price tags, when compared with the disk. They are also polarized, as shown by the polarity marking on the component in Fig. 5-20C. (This would be a 10 μ F / 20 volt component: value first, then voltage rating.) The reason for their high-grade ratings is not high

tolerance alone, but their reliability. If they are operated within their ratings, they are guaranteed not to break down. This is very important in sensitive digital circuits where they are used extensively.

Adding capacitors is easy. Just remember that the rules for adding serial and parallel configurations is just the opposite when compared with the rules for the resistor. Figure 5-21A shows that the equivalent capacitance for *series* capacitors is found by adding the reciprocals. The equivalent capacitance of capacitors in *parallel* arrangement is found by simpler linear addition of the individual values, as in Fig. 5-21B.

Now a quick rundown of some applications.

Power Supply Filter Capacitors. Figure 5-22A shows a circuit powered by an unregulated dc supply. The load (circuit consuming the power) is represented by a resistance, R_L . The ripple in the source voltage is undesirable, and must be removed. Figure 5-22B illustrates an exaggerated ripple of +3 to +8 volts, varying about 5 volts. The ripple is filtered out by placing a very large electrolytic can across the load with the polarity as shown. The large can will charge and discharge

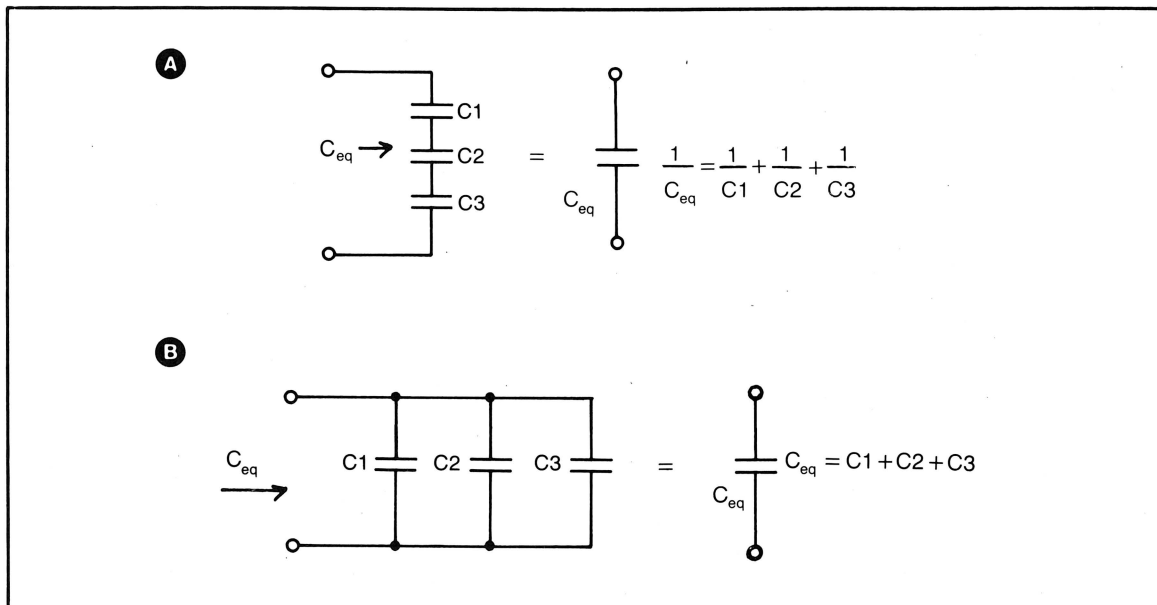


Fig. 5-21. Figuring the equivalent capacitance of (A) series, and (B) parallel capacitors.

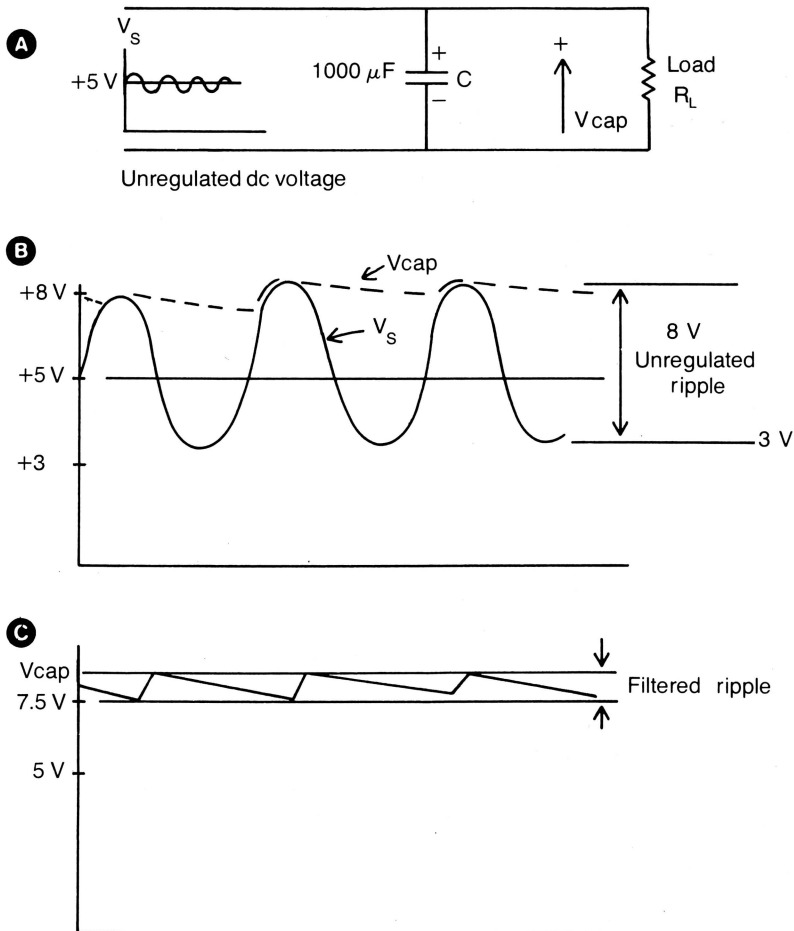


Fig. 5-22. Filtering ripple. (A) Dc power source with ac ripple, and filter capacitor in place. (B) Effect of large filter capacitor. V_{cap} rides on V_s and decays slowly. (C) The variation in the voltage applied across the load has been reduced from 100% to about 7%. See text.

slowly. It will charge to the peak voltage, 8 V, and because of the large TC (large capacitance) it will decline very little from that level. The capacitor filters out the ripple by storing and delivering charge by virtue of its large capacitance, hence the filtered voltage will look something like that in Fig. 5-22C. The ripple has been reduced from 100% ($8 - 3/5$) to about 7% ($8 - 7.5 / 7.5$)!

Naturally, there are formulas that tell you what size capacitor should be used for a certain size load and desired reduction in ripple, if you are designing

a power supply. Also, you are left with a roughly 7.5 V average voltage in this example. This can be reduced to the desired level (5 V, 3 V, etc.) by means of regulators or zener diodes, which we will discuss in the next chapter.

Dc Blocking Capacitors. Figure 5-23A shows an audio signal from the output amplifier stage of, say, a radio. The signal *rides* on top of a 10 volt dc level, one which results from bias or power supply voltages within the amplifier stage. This dc voltage, if applied to an output speaker would result

in distorted speech and music, or no output at all. (The speaker baffle would be pinned or restricted in position by this dc level.) In Fig. 5-23B, the dc voltage is *blocked* by the small $10\ \mu\text{F}$ capacitor. Only the audio voltage, which fluctuates at a frequency of between a few hundred and several thousand Hz, will pass the capacitor.

For instance, at a 1 kHz (typical mid-range audio) frequency, this capacitor would have a reactance of $1/6.28\ (10^{-5}\text{F})\ (10^3\ \text{Hz})$ or about 16 ohms. This is a low resistance for current flow, and so audio frequencies are readily passed, but dc signals are blocked. The actual signal component that appears at the speaker is only the ac component of V_{in} . Current flow through the speaker is shown to the right of the circuit, with the dc “contaminant” removed.

Despiking Capacitors. Figure 5-24C illustrates a typical situation in a digital system, namely, a set of digital circuits supplied by a power source or supply with power and ground lines common to each major unit of the system. The supply

voltage is a regulated, constant value of +5 volts. However, the total current supplied to the circuit is not a simple straight line. It is interrupted by occasional *current transients*, pulses or spikes, one of which is shown in the figure.

These current transients come from two sources. They can be caused by switching noise in the digital circuits, which are transmitted throughout the entire system via the power and ground lines which are common to the entire system. Or, power spikes may also be generated from outside the system: from the power supply itself, from electrical machinery in the same building or from irregularities in the utility line. Power spikes are trouble; they can cause erroneous data, and can sometimes even *lock up* the system.

The solution to current transients is obvious from the diagram. A bypass capacitor conducts high frequency spikes to ground. Even a small capacitor, say $.1\ \mu\text{F}$ or so, will conduct these spikes to ground before they reach the rest of the system. In this example, we assume that the noise comes from

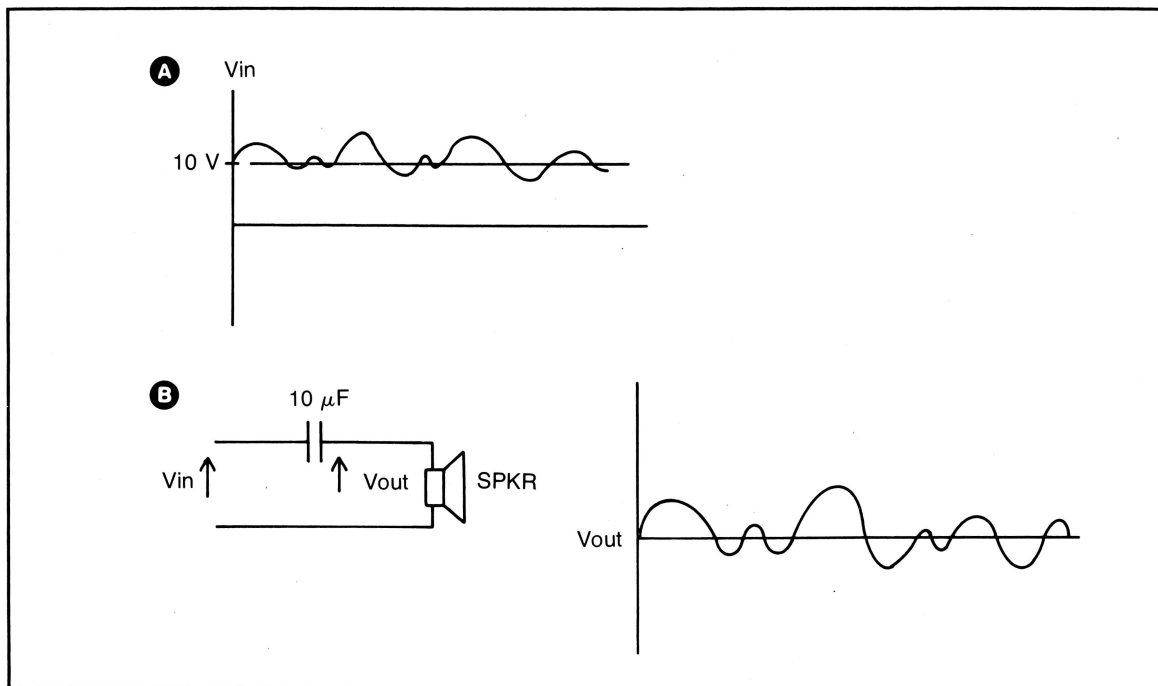


Fig. 5-23. Dc Blocking. (A) Audio waveform riding an undesirable dc level. (B) Removal of dc component by a small blocking capacitor.

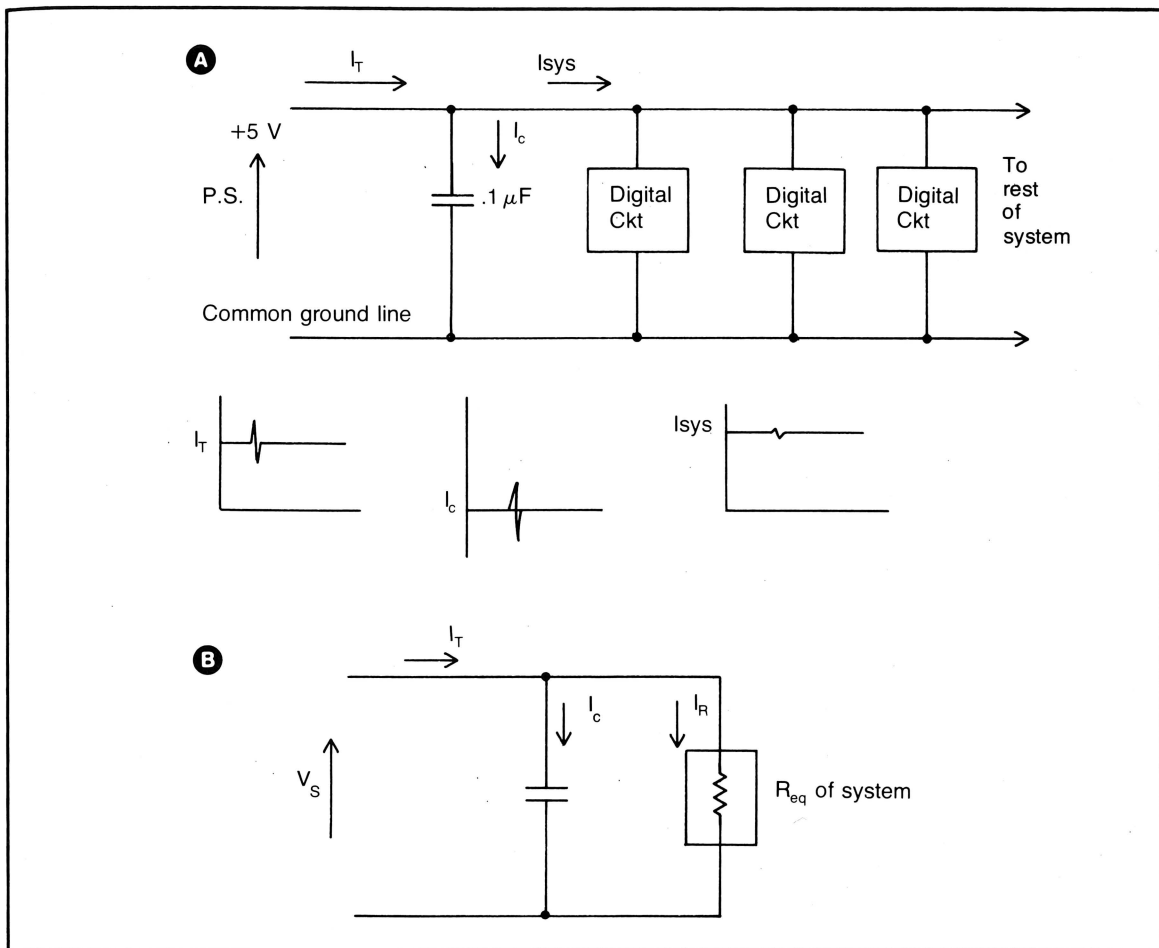


Fig. 5-24. Removing current spikes. (A) Propagation of a high frequency current spike. Here the power supply is the source of the troublesome spike. The low reactance of even a small capacitance to very high frequencies allows most of the spike current to pass through it to ground. Transients originating in the digital circuits would likewise be dissipated. (B) Equivalent circuit of the system.

outside the circuit, though spike generation from TTL switching phenomena is just as likely. As you can see, the spike on the supply current, I_T , is divided into system current (I_{sys}) and despiking capacitor current (I_c). I_{sys} is very small since most of the transient goes through the bypass capacitor.

In Fig. 5-24B the system is redrawn, with the system represented by an equivalent load resistance, R_{eq} . This setup is really the same as the one for the power supply filter capacitor just discussed. The only difference is that the current is smoothed or filtered of very high frequency variations,

therefore you need only a very small capacitor to do the job.

Despiking capacitors must never fail. If they short out, then the power supply shorts out and the current surges can be disastrous. Needless to say, digital circuits and components must have high-grade never fail, capacitors. Therefore, high reliability tantalums are used.

RC Time Constant. The RC time constant is utilized in clocks or oscillators. Figure 5-25A illustrates one of the simplest and cheapest oscillators you can construct. It consists of two hex inverters,

a resistor and a capacitor. The output is a square wave.

Assume that the output of inverter 2 is high, as indicated by the arrow on the output wave. Then the output of inverter 1 will be low. The capacitor will charge through resistor R with polarity and direction of positive current flow as shown. When the charging current through R decreases (exponential decay) the voltage drop across R (I_R) will eventually reach a logic low. This low will be present on the input of inverter 1, and its output will be high, making the final output of inverter 2 low.

Charging current then flows through R in the reverse direction. When the voltage on C reaches a logic high (reversed polarity to that in the figure) the input of inverter 1 will be high and the output of inverter 2 will be high, back where we started. The time constant for this oscillation is proportional to

RC. For this configuration the actual value of TC is about $\frac{1}{2} RC$. The frequency is given as $F = 1/6.28 RC$.

The RC Time Constant in Measurement.

As a final example of a circuit employing the RC time constant, refer to Fig. 5-25B. A variable resistor connects a capacitor to a +5 V source. The capacitor can be momentarily grounded and discharged to 0 V under the control of a computer. At a set time, the switch indicated in the diagram first closes to discharge the capacitor, then opens, allowing it to charge through the resistor. At this time a counting loop in software begins. The capacitor then begins recharging through the variable resistor.

The time it takes the capacitor to charge up and reach a TTL input logic high (2.0 V) is determined by the TC, a product of the capacitor and the vari-

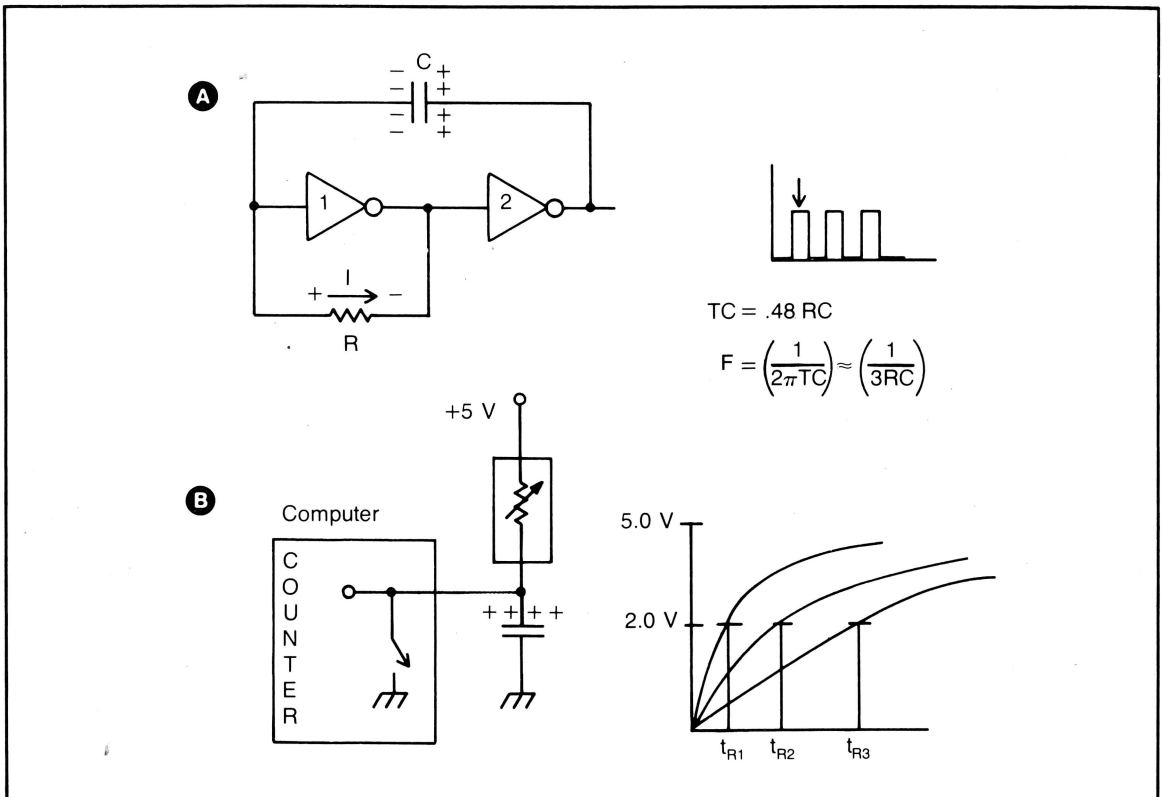


Fig. 5-25. (A) Hex oscillator, the frequency of which depends upon the values of R and C. (B) Using the RC time constant for measurement of physical quantities. See text for discussion of both circuits.

able resistor. When logic high is reached, the counter stops, and the value of the count is proportional to the value of the variable resistor. As shown in the family of curves to the right of this circuit, the time constant varies as the resistor value varies. The various times— t_{R1} , etc.—are proportional to the resistor value and to the specific count in the timing program.

So by measuring a time constant, you measure, in this case, a resistance. Now, what varies the resistance? Here are a few causes.

- ☐ Position. Rotating a rheostat (variable resistor) changes its value. This is the principle of the game paddle and joystick.

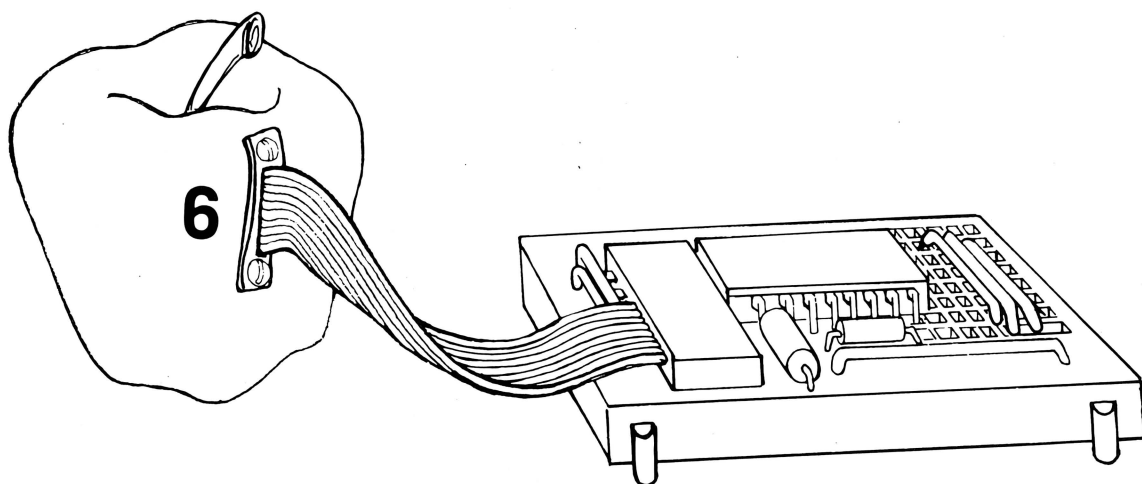
- ☐ Temperature. Thermistors are resistors with special thermal properties. Their resistance changes considerably with temperature changes.

The count is then proportional to temperature!

- ☐ Light. Resistive photocells, the resistance of which varies with light intensity, are used. The time constant, and therefore the count, is proportional to intensity of light falling on the photocell.

- ☐ Pressure, pH, chemical concentration and other physical properties are additional examples. If you can cause a change in resistance in response to changes in the physical quantity under study, you can measure that quantity by such pure resistive transducers as the one in Fig. 5-25B.

NOTE: The actual technique of measurement varies with the computer. In the case of Apple, the counter measures the duration of a square wave. This duration is determined by the simple RC time constant. Details of this are presented in Chapter 11.



The Diode and Transistor

Like the last chapter, this one is a mix of basic theory that you must know as a bare minimum, combined with practical settings in which those principles apply. You'll first learn about the simplest semiconductor device, the diode, and see how the concepts of resistance and capacitance can be used to describe both its normal operation and its limitations under extreme conditions. Transistor action is covered next. Transistor fundamentals could span several chapters and involve lots of mathematics. Fortunately, we're interested in only the rudiments of transistor operation. Because we are interested in the transistor as a digital device, our treatment is simplified. You'll learn its properties as a *transfer resistor*, how it can be used as a switch, its configurations as a current source and current sink, and some of its limitations in terms of speed and driving power.

All of this will make the nuts and bolts coverage of TTL circuit operation and technical specifications in the next chapter much easier, and it will give you an intuitive understanding of the practical electronics of digital circuitry.

SEMICONDUCTOR MATERIALS

Semiconductor material is manufactured using silicon (Si) as a base substance or *substrate*. As with the atoms of every element, the silicon atom has a nucleus, one or more inner electron shells and an outer or *valence* shell. It is this valence shell, or more accurately the number of electrons in it, which determines its chemical and electrical properties.

The valence shells of the elements we will be considering have room for eight electrons. A full valence shell contains eight electrons. And it is the most stable configuration for any element; it is the lowest energy state or most desirable state for an atom. One way an atom can attain a full valence shell is by sharing electrons with other atoms by forming chemical bonds.

An Si atom is represented in Fig. 6-1A with nucleus, inner electron shells and valence shell. The valence shell contains four electrons and needs four more for a complete shell. We can say that the Si atom has four electrons available for sharing, or bonding, and that it has a valence of 4. This is more

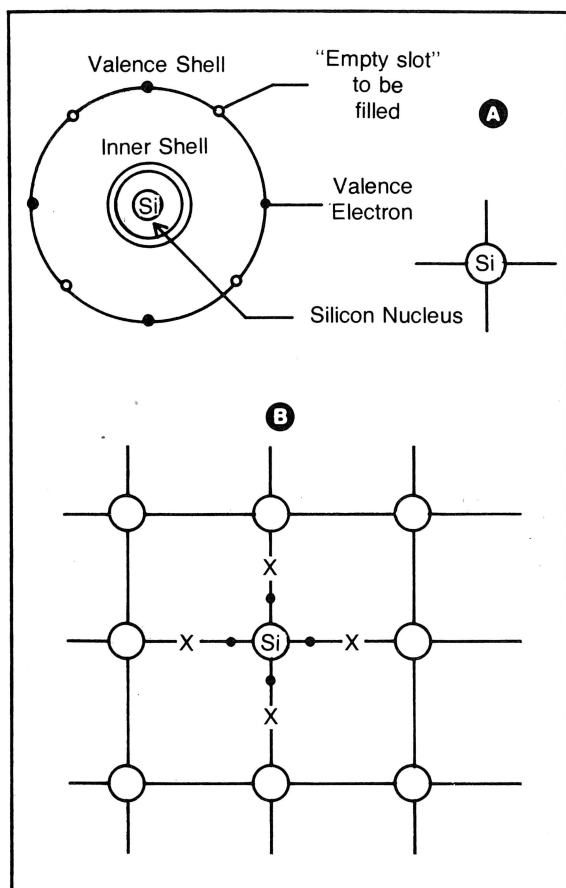


Fig. 6-1. A) Silicon has four valence electrons and four empty slots. An Si atom can form four covalent (sharing bonds) with other atoms. Ball and stick representation is to the right. B) A pure silicon crystal. The central atom shares its electrons (dots) with those from neighboring atoms (x's), for a total of four bonds.

conveniently represented as a ball-and-stick figure, shown to the right of the shell figure.

By borrowing the electrons from the outer shells of four other silicon atoms, a particular Si atom can assume a lower energy state. True, it must share each of its *own* outer electrons with the other atoms, but this sharing does allow a complement of eight electrons to be present on the outer shell at least part of the time. The stable configuration that results is a *crystal* of silicon, shown in Fig. 6-1B.

It is important to note that the Si atoms clutch their shared electrons closely. There are no spare

electrons to float around, and there are no positive areas, or holes, where electrons are absent. There are no free + or - charge carriers. This condition makes the Si crystal an electrical insulator.

Now, silicon can be transformed into a semiconductor by the addition of elements with a different valence. The addition of even very small amounts of such materials such as gallium or arsenic will result in what you might call *odd fits* in the Si crystal. This *doping* process leaves extra + or - regions in the crystal which can act as charge carriers. Doping does not give silicon the conductivity of copper wire, but rather imparts a partial (semi-) conductive capability. Let's see how this comes about.

N-Type Material

Arsenic (As) has five valence electrons; it is pentavalent. Note the ball and stick figure in Fig. 6-2A. When a tiny amount of arsenic is added to tetravalent (4 electrons) silicon, there is a small surplus of unpaired electrons in the doped crystal because the fifth valence electron on As is not part of a bond pair. The crystal is still electrically neutral; however these unpaired electrons give a net concentration of negative charge wherever As atoms occur. These electrons are relatively *mobile*, compared with the other electrons, and can move under the influence of an applied voltage. These mobile electrons are called the *majority charge carriers*. There is also a very small population of *minority carriers* in this n-type semiconductor material which result from imperfections and impurities in the silicon. These impurities are not shown in this figure. These are called holes.

P-Type Material

Gallium (Ga) is a trivalent (three valence electrons) substance used to dope silicon to form p-type semiconductor material. In this case, there is also an unpaired electron. It resides on the Si atom just below the Ga atom in Fig. 6-2B. As with N material, the crystal is electrically neutral. However, the *charge distribution* in this region is relatively positive, because the incomplete shell on the Si atom does not completely "cover" or hide the

positively charged Si nucleus. The result is a net concentration of positive areas or *holes*. These holes are the *majority carriers* in p-type material and are free to move under the influence of an applied voltage. A small population of minority carriers also exist in p-type material due to crystal imperfections and impurities.

Current Flow

Current flow through p-type and n-type material is shown in Fig. 6-3. If a battery source is connected to an n-type semiconductor, as in Fig. 6-3A, the mobile majority carrier electrons will migrate through the crystal. They actually jump from one atom to the next, momentarily ionizing

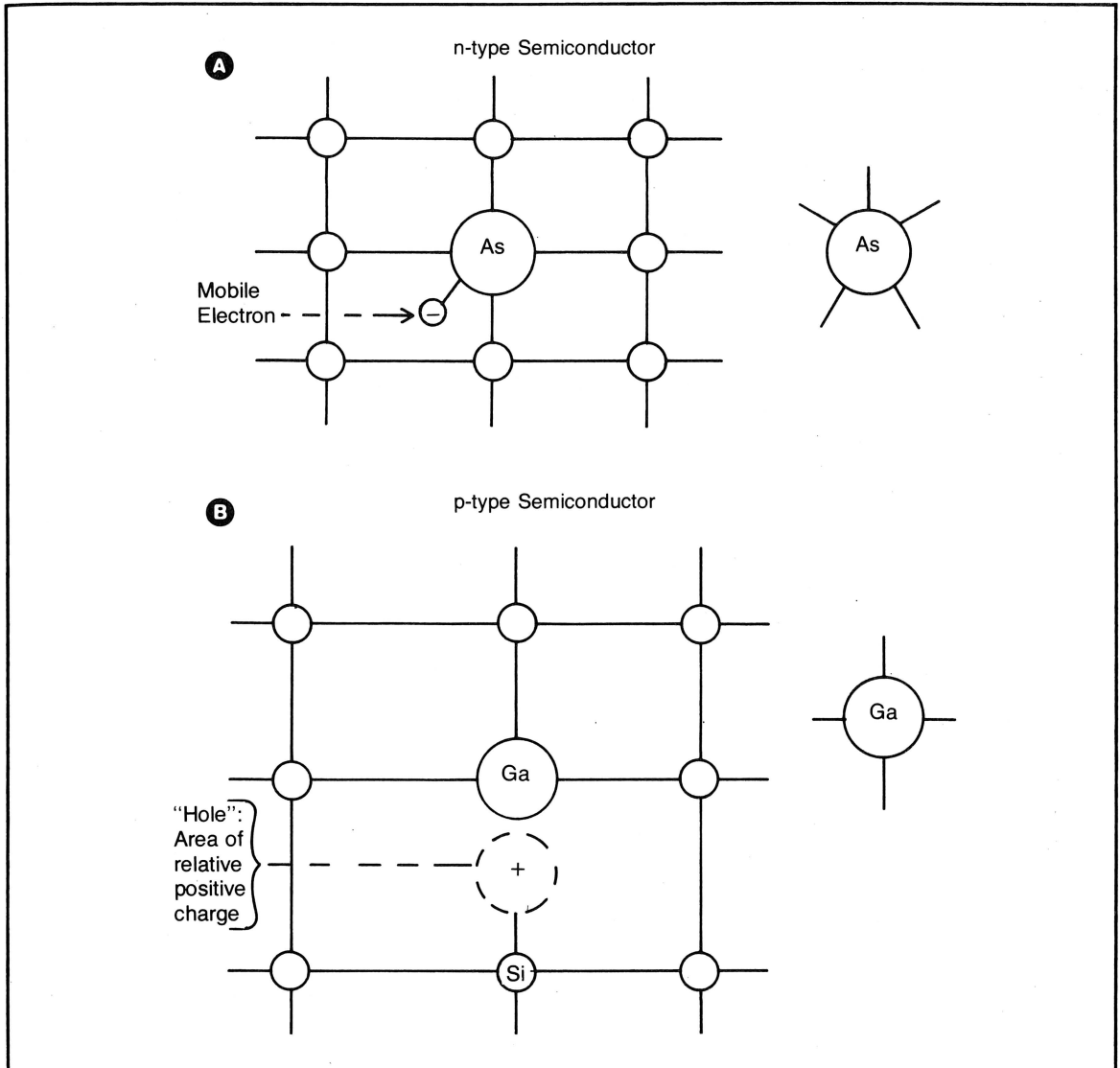


Fig. 6-2. A) A crystal of impure Si is doped with pentavalent arsenic (As). A region of relative negative charge exists due to the unpaired electron from As. This relatively mobile electron is a potential charge carrier in this n-type material. B) A crystal of impure Si is doped with trivalent gallium (Ga). The charge carrier in this p-type material is a region of relative positive charge, or hole.

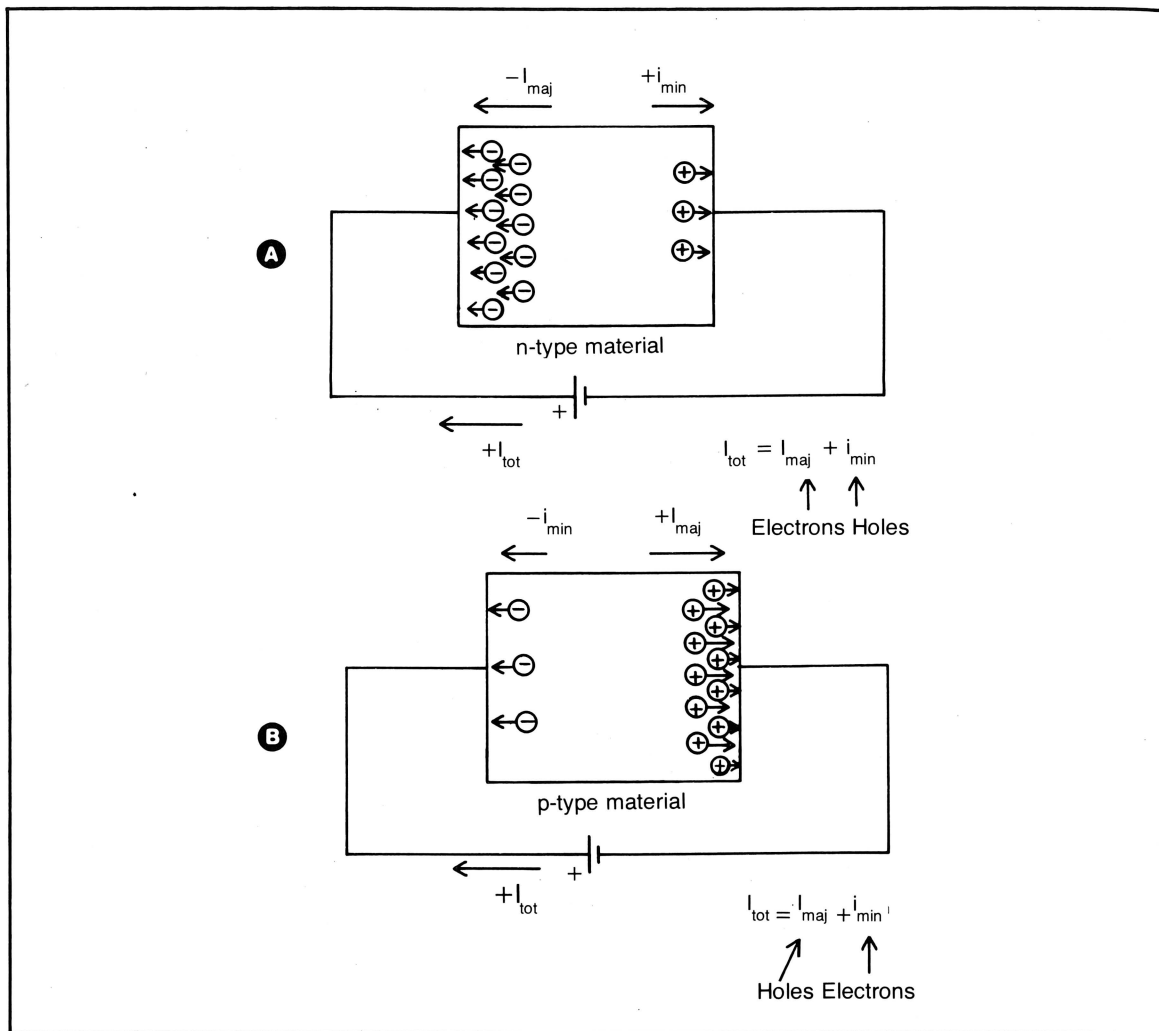


Fig. 6-3. A) Under an applied voltage, the total current flow in n-material is the sum of majority carriers (electrons) and minority carriers (holes) moving in opposite directions. Net current flow is represented by positive current convention. B) In p-material, the net current flow is the sum of majority carriers (holes) and minority carriers (electrons), also moving in opposite directions in this doped crystal.

that atom with a negative charge, and then moving to the next atom, towards the positive terminal. A few minority carrier holes also move, but in the opposite direction, towards the negative battery terminal.

Regardless of the physics of charge carrier movement within the semiconductor, we can represent net current flow in the circuit by means of a current, $+I_{total}$, using the positive current flow convention. This current is the sum of the majority and

minority currents within the crystal. Most of this circuit current is due to the majority carriers, I_{maj} , with a small contribution from the minority carriers, i_{min} . Its magnitude is limited by the resistance of the semiconductor, a resistance that might range from several hundreds to several thousand ohms.

Current flow in a p-type semiconductor is illustrated in Fig. 6-3B. Here, the majority carriers are holes, and the minority carriers are electrons. These charge carriers jump from one atom to the

next as they migrate towards their respective battery terminal. As with the n material, the charge carriers momentarily contribute a net excess of positive or negative charge to the atom on which they are briefly sitting. This atom is momentarily ionized before they move on. Hole current, represented by I_{maj} , makes up most of the total current flow in this p-material, with a small contribution from minority carrier electron flow, i_{min} . Total current is again represented in positive flow convention. (This convention is implied in this book unless otherwise stated).

THE DIODE

The semiconductor diode is essentially a pn junction. It is formed by butting p material against n material and attaching leads to either end, as in Fig. 6-4A. Knowing about pn junction behavior is essential to understanding more complex semicon-

ductor devices, such as the transistor, and understanding semiconductor circuits, both discrete and integrated. The concepts that you must be familiar with are:

- ☐ The *junction barrier voltage* and forward and reverse *bias*.
- ☐ The *characteristic curve* for this device.
- ☐ The existence and implications of *internal capacitance*.

Each of these three aspects of the diode will be examined in the next three sections.

Diode Characteristics

When p- and n-material is butted together to form a pn junction, as in Fig. 6-4A, a *barrier voltage* across the junction results. The reason is as follows:

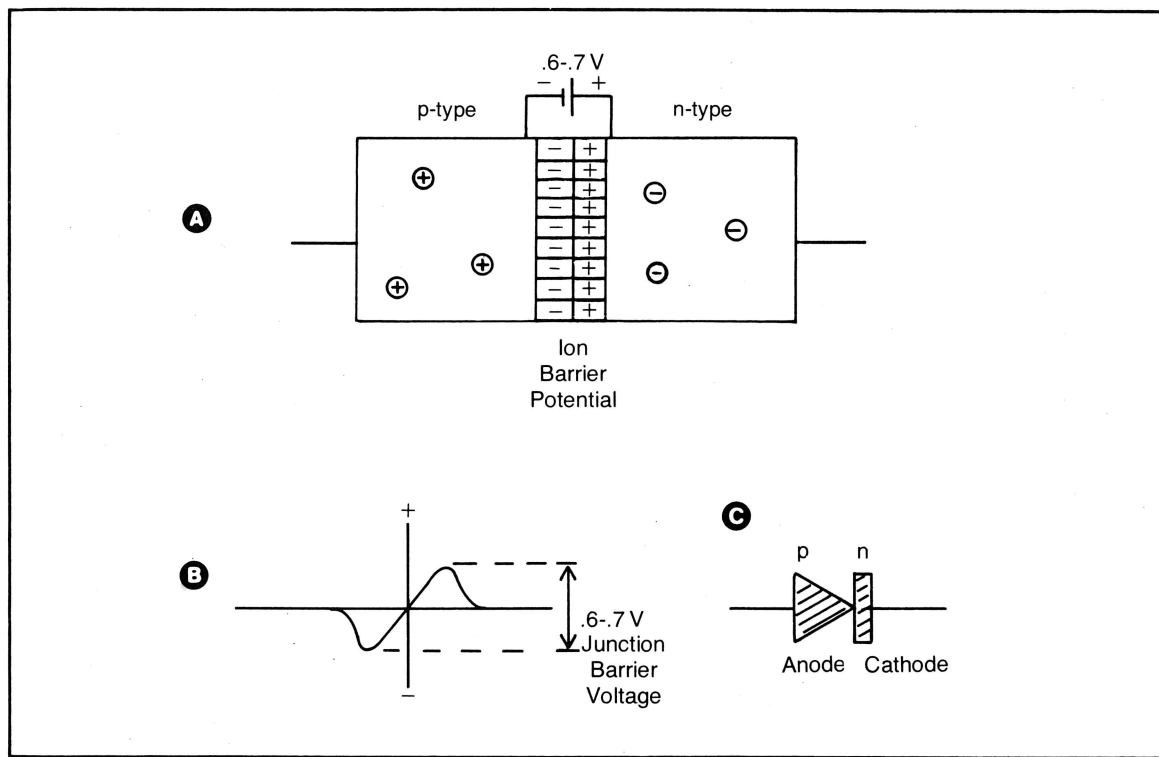


Fig. 6-4. A) The unbiased (no applied voltage) Silicon based pn junction has a barrier voltage of .6-.7 volts. B) Graph of barrier voltage, where potential is plotted against distance on either side of the junction. C) Schematic symbol of the semiconductor diode.

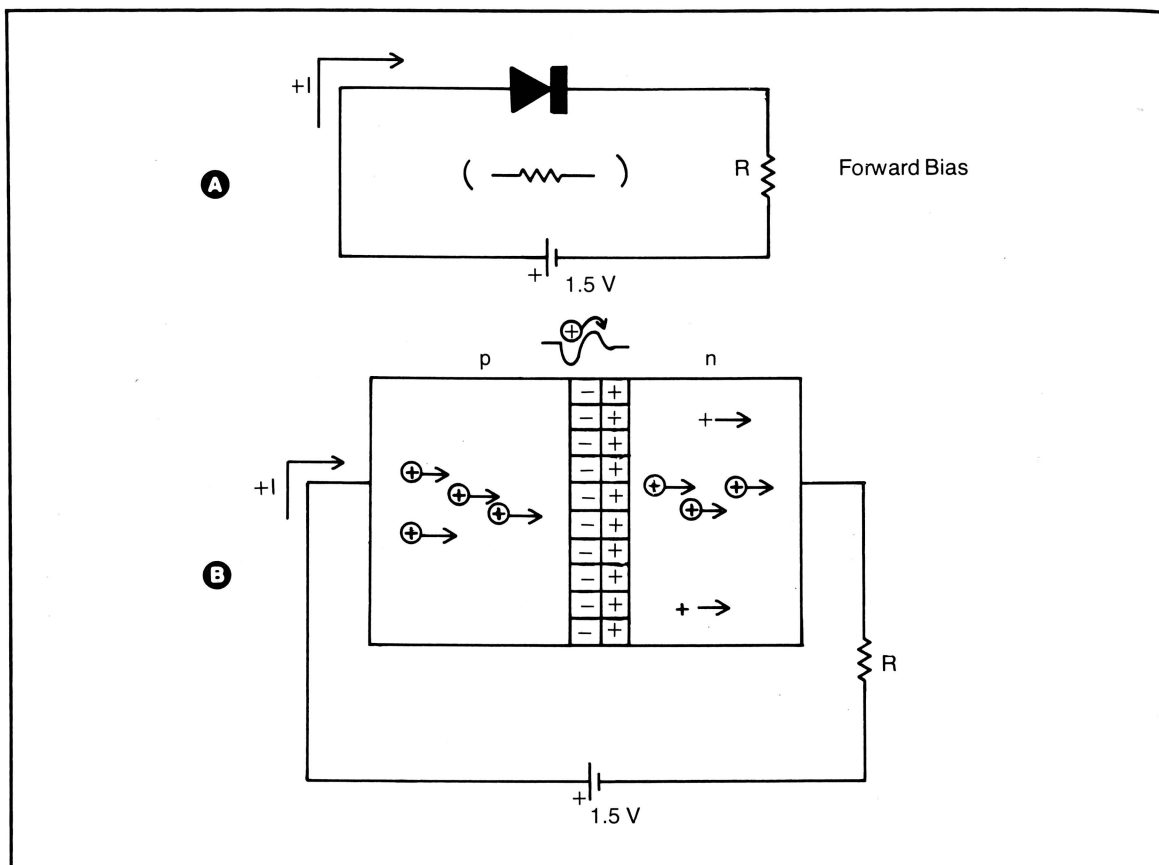


Fig. 6-5. A) The forward-biased diode is like a low value resistor. B) Positive current flow representation of net current flow under an applied voltage. This current is the sum of majority holes from p-material, plus a small contribution of minority holes from the n-material.

Mobile electrons in the n-type semiconductor are attracted to the regions of relative positive charge—holes—in the p-type semiconductor. These electrons cross the junction and sit in these positive areas, in this case in the valence shell of silicon (refer back to Fig. 6-2B). The p-side Si atoms are negatively ionized. Further, because of this migration of electrons from the n-type semiconductor, the N side of the junction has a real deficit of electrons and is no longer electrically neutral. There is a net positive charge on the n side. One would express this by saying that the arsenic atoms that lost their electrons due to migration to the p material are positively ionized. The ionized atoms near the junction are represented by the boxes containing + and - signs in the figure.

There is a limit to this migration of charge because as the charge builds up on either side it tends to repel further migrating charges coming in from the opposite side. The net result of this charge accumulation across a junction is, as you would predict, a voltage potential. It is referred to as the *junction barrier voltage*. For the case of silicon pn junctions, this voltage has a magnitude of 0.6 to 0.7 volts. The *polarity* of this junction potential is shown in Fig. 6-4. The n side is relatively more positive than the p side because of ionization of the atoms on either side. This situation is represented by the small battery symbol in 6-4A, and by the graph in 6-4B. The significance of the barrier potential will be clear in a moment.

The symbol of the diode is given in Fig. 6-4C.

The p-type material, signified by an arrowhead, is called the anode. The n-type material, signified by the bar, is called the cathode.

If we now apply a voltage to the diode, so that the positive and negative terminals are attached to the p and n sides (anode and cathode), then the diode is said to be *forward-biased*. In this instance the applied voltage is 1.5 volts. This voltage is sufficient to overcome the barrier voltage, and current will flow through the circuit in the direction indicated in Fig. 6-5A. In this forward-biased state, the diode presents very little resistance to current flow, as suggested by the small resistor symbol below the device. Resistor R is necessary to limit current flow.

Figure 6-5B illustrates what is going on inside

the diode. Holes are injected by the battery and are forced up against the barrier potential. Once the holes are through the barrier, they are attracted towards the negative battery terminal and continue their passage. Because the applied voltage is sufficient to overcome this barrier of about 0.7 volts, current I flows. Its magnitude is limited by resistor R. In this figure the majority carriers from the p semiconductor are represented by the circled + signs. The minority carriers in the n semiconductor, represented by the uncircled + signs, also contribute a tiny bit to current flow in this forward-biased state.

Note that I could have represented the flow of *negative* charge carriers—majority electrons from the n material and minority electrons from the p

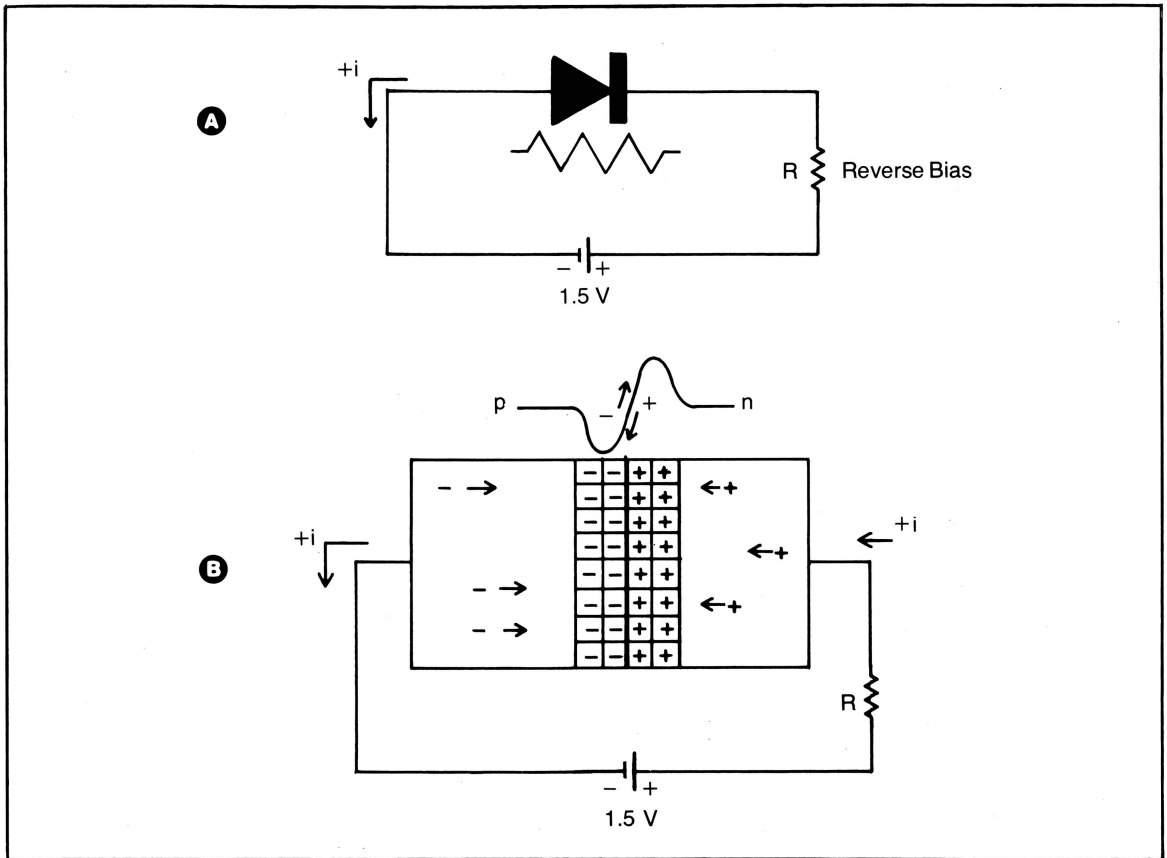


Fig. 6-6. A) The reverse-biased diode is like a high value resistor. B) Essentially, only minority carriers flow in this diode. Majority carriers cannot move against the junction potential, and in fact are trapped in the barrier. The barrier grows in width and height with increasing external voltage.

material—as going in the opposite direction. This actually occurs, but was omitted from the figure for simplicity).

What voltage value would you measure across this forward-biased diode? You would measure a value around 0.7 volts. This is the potential which is dropped before significant current flow occurs. This forward bias voltage remains fairly constant even if we increase the applied voltage significantly.

The case of the *reverse-biased* diode is illustrated in Fig. 6-6. Here, the positive and negative terminals are attached to the n and p sides of the diode, respectively.

Almost no current flow occurs because the applied voltage is in the direction to *increase* the barrier voltage. The reason for this is that the positive terminal injects holes into the n material, which then migrate towards the junction. There, they ionize the As atoms. That is, positive charges accumulate on the n side of the junction. The net charge accumulation is much greater than that occurring in the unbiased diode. Therefore the barrier voltage in this reverse-biased diode is much greater. This is illustrated in Fig. 6-6B by the increased width of the zone of ionization and by the higher and steeper barrier potential curve above the device.

A similar process of electron injection by the negative terminal, with resultant negative charge accumulation on the N side, also occurs.

The result is that there is a greater number of ions on either side of the junction. Both the magnitude of the barrier voltage and the width of the barrier region are relatively large.

Only a tiny current flow in this situation. It is made up of the positive minority carriers in the n-material which fall down the barrier potential, as indicated in the curve. (Minority electrons from the p-material also flow in the opposite direction. We use positive current convention, however, for consistency.) This current is on the order of microamperes.

If you measured the voltage drop across the diode, you would find a value of very nearly 1.5 volts! This is easy to understand if you think of the

diode as being extremely resistive to current flow, thanks to the elevated barrier voltage and the few minority charge carriers available. Since the effective reverse-biased resistance is very large (megohm range), most of the voltage drop would occur across the diode, even if resistor R were tens or hundreds of thousands of ohms.

As the reverse-bias voltage is increased, the opposing barrier voltage increases with it and very little additional current flows. Again, it is as if the reverse-biased diode were a very large resistor, approaching an open circuit in the ideal case. This is suggested by the large resistor symbol in Fig. 6-6A.

The Diode Characteristic Curve

We can summarize what has been said in the form of the so-called *characteristic curve* for the semiconductor diode. Figure 6-7 depicts such a characteristic curve for a typical general purpose diode. The voltage drop across the diode, on the horizontal or x-axis, is plotted against the current through it, on the vertical or y-axis. Positive voltage, to the right, represents forward bias. Negative voltage, to the left, represents reverse bias.

Starting from zero volts, we increase the forward bias. Current flow will be negligible until the applied voltage reaches about 0.7 volts. At this value, the junction barrier potential is exceeded, and significant current flow occurs. The diode is conducting in a fully forward-biased condition. Because a minimum voltage of about 0.7 V is required for current flow, this value of applied voltage is called the forward bias voltage. In any event, it is about the same magnitude as the junction barrier potential.

The voltage drop across the diode will increase a little as more current flows in response to increases in applied voltage. This is because injected charge carriers do not accumulate at the junction. Rather, once they are pushed over the barrier by the repelling terminal, and they are accelerated to the attracting terminal on the other side. Therefore, the barrier voltage increases only slightly as applied voltage is increased. This is why

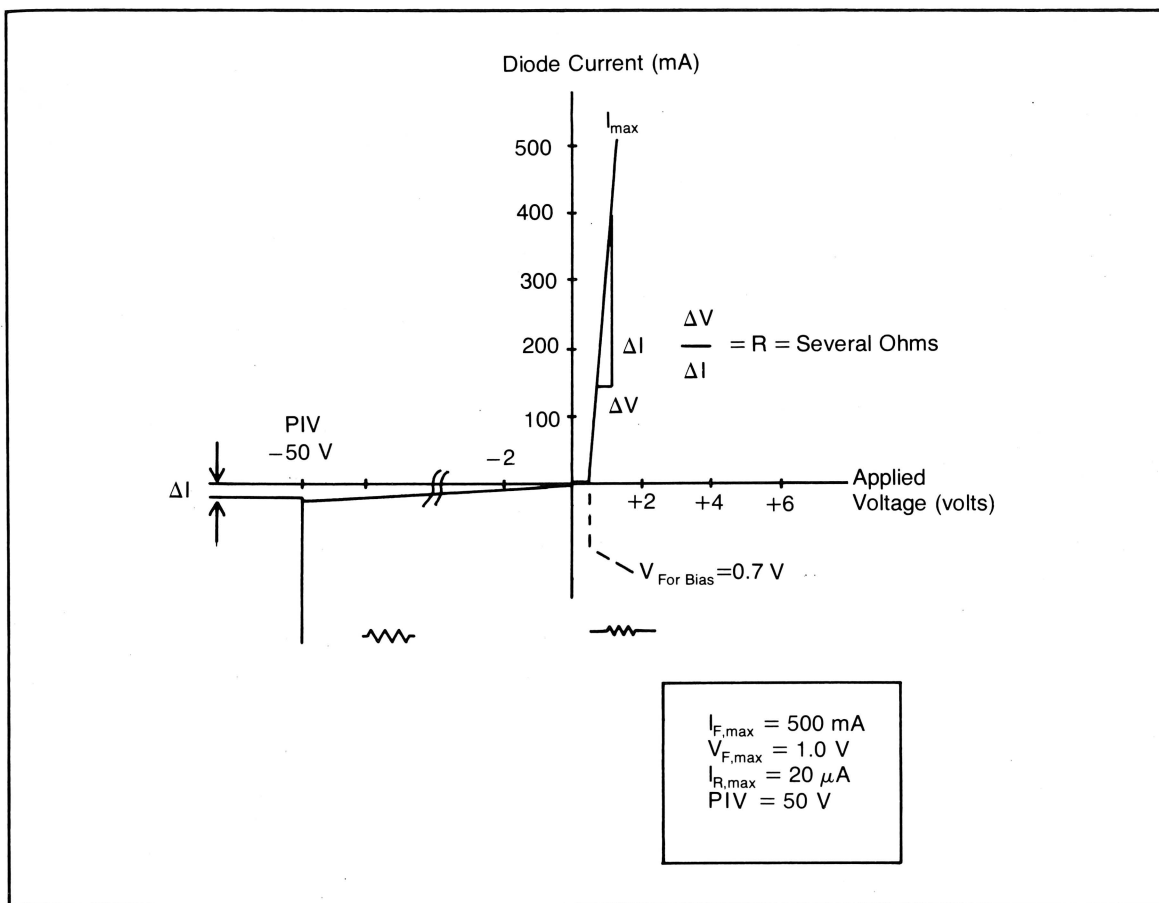


Fig. 6-7. The characteristic curve for a general purpose diode. See text for discussion.

the forward-biased junction presents a relatively small resistance to current flow. Just what is the forward resistance of the diode? It is the *slope* of the forward characteristic line, the so-called dynamic part of the curve. As you can see from the figure, we can calculate the dynamic or conducting resistance of the diode by taking the inverse of the slope of this line: $\Delta V/\Delta I$. This resistance is quite low, a few ohms or a few tens of ohms. This is suggested by the small resistor symbol below this half of the characteristic curve.

To the left is the *reverse-biased* portion of the characteristic curve. In this state, the diode is essentially nonconducting because of the large barrier potential and the negligible contribution of minority carriers to current flow. The very high reverse bias

resistance is evident from the very shallow slope of this part of the curve: the change in current for a given change in voltage is very small. That is, $\Delta V/\Delta I$ is large and approaches infinity in the ideal diode. The high reverse bias resistance is suggested by the large resistor symbol below this half of the curve. Typical values are in the range of several hundred kohms to several megohms.

The ratings associated with *forward* bias conditions are the maximum forward current, $I_{F,max}$, and the corresponding maximum forward bias voltage drop, $V_{F,max}$. Values of 500 mA and 1.0 volts are typical of small signal, general purpose diodes.

As the absolute value of the applied *reverse* voltage is increased in the negative direction, the voltage drop across the diode also increases. At

some value of reverse voltage—called the *peak inverse voltage* (PIV)—the diode breaks down. Beyond this PIV, any further increase in applied voltage causes large changes in reverse current (current flowing in the direction opposite that of forward current). Right at the point of breakdown, at the “knee” of the reverse characteristic as it were, is a current known as the maximum reverse current, $I_{R,max}$. Typical values for the PIV and $I_{R,max}$ ratings might be 50 volts and 20 μA . (It is understood that PIV is a negative voltage relative to the polarity for forward bias. Also, power diodes have much higher PIVs.)

Pn Junction Capacitance

The third item on the list of diode essentials mentioned earlier is that of pn junction capacitance. The *internal capacitance* of the diode is the only important property not expressed in the diode characteristic curve, a property that deserves special attention.

The basis for junction capacitance can be understood intuitively. Because of the barrier voltage, the junction itself constitutes a relative impasse to current flow. It acts in the same way as the dielectric does in a capacitor. In a similar manner, the mobile carriers on either side of the junction act much as the + and – charges do on either side of the capacitor plates. This configuration of an ion barrier and mobile carriers on either side imparts a capacitive property to the diode.

Further, the value of this internal diode capacitance, while quite small (picofarad range), varies depending on whether the diode is forward or reverse-biased. When forward-biased, there are many majority carriers on either side of a relatively thin barrier region. But when the diode is reverse-biased, there are very few majority carriers (there is no current flowing) and the barrier region is relatively broad. The effective diode capacitance is therefore greater when it is forward-biased than when it is reverse-biased.

The implications of diode internal capacitance are clear when you examine the circuit of Fig. 6-8A. In this schematic, the *equivalent circuit* for the diode is represented by the resistance, R_d , and the

capacitance, C_d , in parallel. In order to see the effect of diode capacitance on a typical digital signal, we apply a square wave as the input and examine the resulting output. The output voltage, V_{out} , is taken off the series resistor, R_s .

In Fig. 6-8B the input and output voltages are compared. A single cycle of V_{in} is shown. When V_{in} is low, the diode is not forward-biased. It is nonconducting or off. When V_{in} goes high, the diode is forward-biased and current begins to flow. Because of the diode capacitance, current rises exponentially and so does the voltage across R_s . It takes a finite amount of time for V_{out} to rise to the level of V_{in} . This time is signified by t_{TLH} , the low to high transition time.

When V_{in} goes low sometime later, V_{out} follows. The decline of V_{out} is an exponential decay because C_d must discharge in the reverse direction. In fact, because of the existence of the mobile charge carriers, the discharge current (which flows in the reverse direction of the charging current) actually causes the voltage across R_s to go somewhat below zero volts. This is the familiar voltage discharge spike you’ve seen before in the capacitor. The time from high to low, t_{THL} , is the transition delay for a high to low transition.

We can say three things about this situation. First, there exists a delay for any semiconductor device, be it diode, transistor, or integrated circuit. This delay imposes a speed limit on the propagation of signals through any semiconductor device. While the effect of this capacitance-related delay is rather exaggerated in the figure, it can be significant in high speed digital circuits.

Second, there are in fact two such delays. One from on (conducting) to off (nonconducting), and one from off to on. There is a greater lag in the on to off state because of the greater effective diode capacitance in the forward-biased condition, as mentioned above. The speed of semiconductor devices is therefore limited by the longer of the two delays, namely t_{HL} .

Third, even the tiny capacitance inherent in semiconductor junctions can cause a small current or voltage spike effect. This is the basis for switching transients in digital circuits, especially TTL-

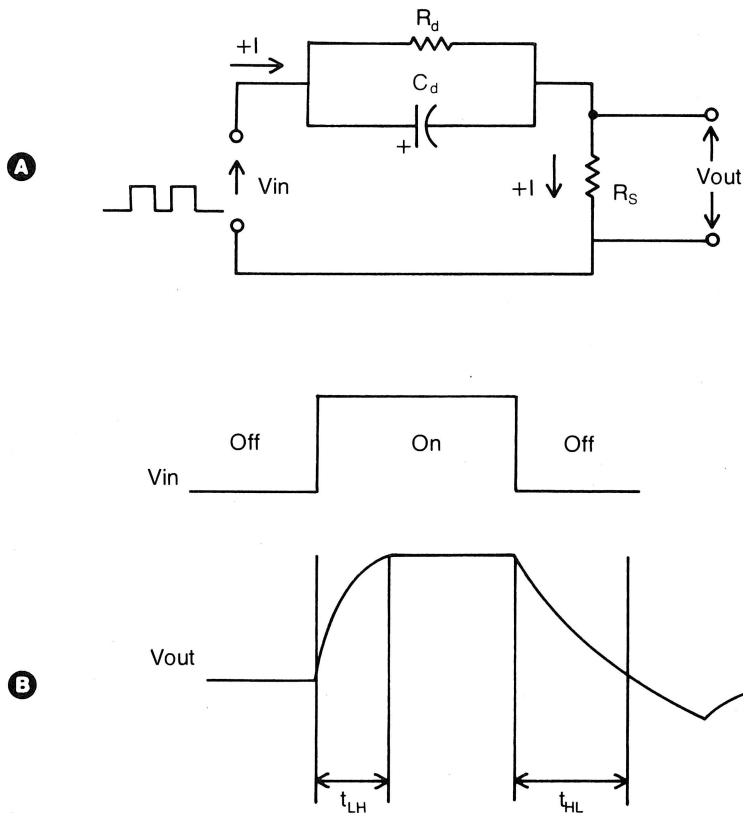


Fig. 6-8. A) Equivalent circuit model of the diode—a resistor and capacitor in parallel. B) An effect of junction capacitance is the lag in the rise and fall of current flow in response to an applied square wave.

based circuits. These transients are more pronounced in the on to off transition, as indicated in the figure, because of the presence of mobile charges in the on state.

Diode Applications

Signal Detection. The typical AM radio signal consists of a radio frequency (rf) carrier on which is superimposed the audio frequency (af) information. This amplitude modulated (AM) signal must be rectified or detected in order for the audio component—music or voice—to be heard. As suggested in Fig. 6-9, the radio signal (1) can be rectified (2) by a general purpose diode. Only the positive part of the signal is conducted. The com-

plementary negative half is not. By filtering out the rf component with a small $10\ \mu\text{F}$ bypass capacitor, the recovered audio signal (3) can then be appropriately amplified and fed to a speaker.

Power Rectification. Similarly, a power rectifier (diode) can be used to convert ac current to a *pulsating dc* current, as indicated in Fig. 6-10, waveforms 1 and 2. The large electrolytic smooths out the voltage and yields a dc voltage with low ripple in waveform 3.

Finally a *zener diode* is used to provide a steady dc voltage, in this case 5.1 V. Zener diodes are used in the reverse biased configuration, as shown in the figure. The zener characteristic curve is given in the inset below the circuit, and as you can see, the

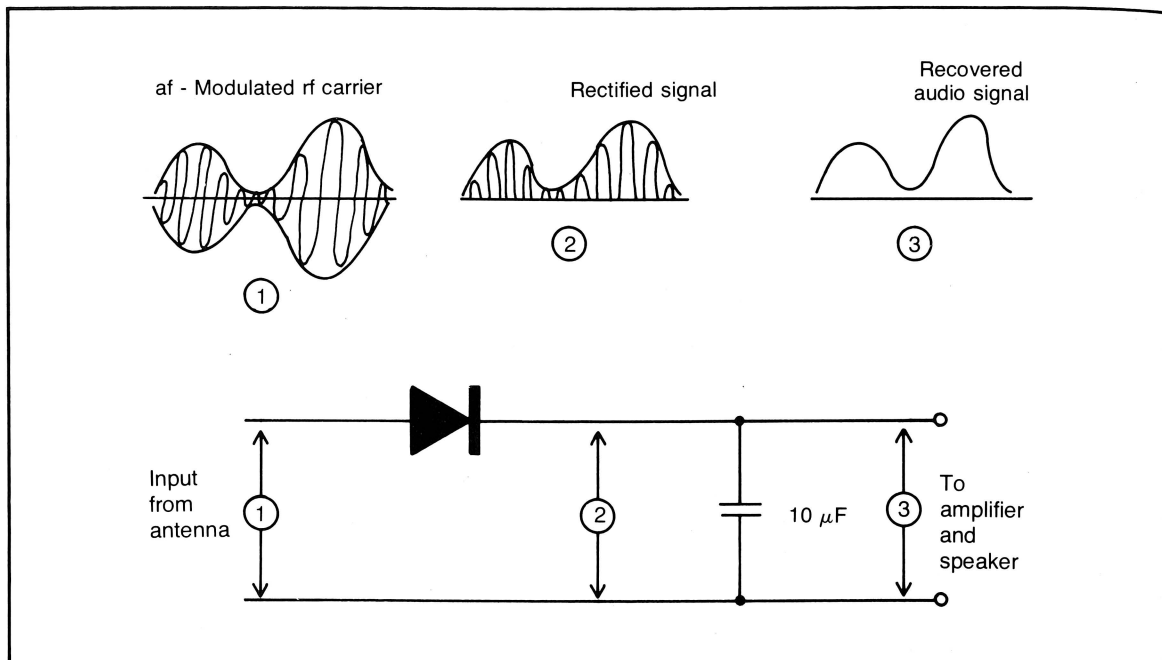


Fig. 6-9. A simple diode detector of the type used in early radios (crystal detectors). Numbered waveforms correspond to the numbered points in the circuit.

breakdown voltage is low, 5.1 V. Provided the applied voltage somewhat exceeds the zener breakdown voltage (V_z), the voltage drop across the zener diode will remain constant, giving a steady dc output. The zener is stable under this breakdown condition because it was designed as such. Of course, their reverse maximum current rating must not be exceeded, and this is the purpose of the current-limiting, series resistor, R_s .

Wave Shaping. This is basically just an extension of rectification. Figure 6-11A thru D illustrate several *series voltage clamp* circuits. In these circuits, output voltage follows input only when current is flowing through the diode and through resistor R . Only the positive part of the sine wave input appears at the output of the circuit in Fig. 6-11A. Note that a forward bias of .7 V is dropped across the conducting diode. The rest of the voltage appears across the resistor, R . Therefore, while the peak (V_p) of the input is 5 V, that of the output will be only 4.3 V. We can also reverse bias the diode, as in Fig. 6-11B. Current will flow only when the input voltage exceeds the 3 volts from the

battery, giving the output shown to the right of the circuit. Figure 6-11 C and D show the waveforms when we reverse the polarities of the diode and battery.

In Fig. 6-12A thru C are shown several *shunt voltage clamp* circuits. In these circuits, the output voltage follows the input only when the diode is nonconducting, or reverse-biased. In Fig. 6-12A the diode is reverse biased during the positive half of the cycle and during that part of the negative half that is more than -0.7 volts. Exactly the reverse is true for the situation in Fig. 6-12B. Both circuits are also known as *clippers* because the top or bottom of the input signal is clipped off at the output. If we now combine the circuits and add reverse biasing using two 3 volt batteries, we have the circuit in Fig. 6-12C. This is called a *corer* circuit because only the central part of the input waveform appears at the output. You'll note that with a sine wave input, the output approximates a square wave.

If bias voltages were variable, as they would be in more elaborate circuits, you can appreciate how the diode can be used to change the sine wave

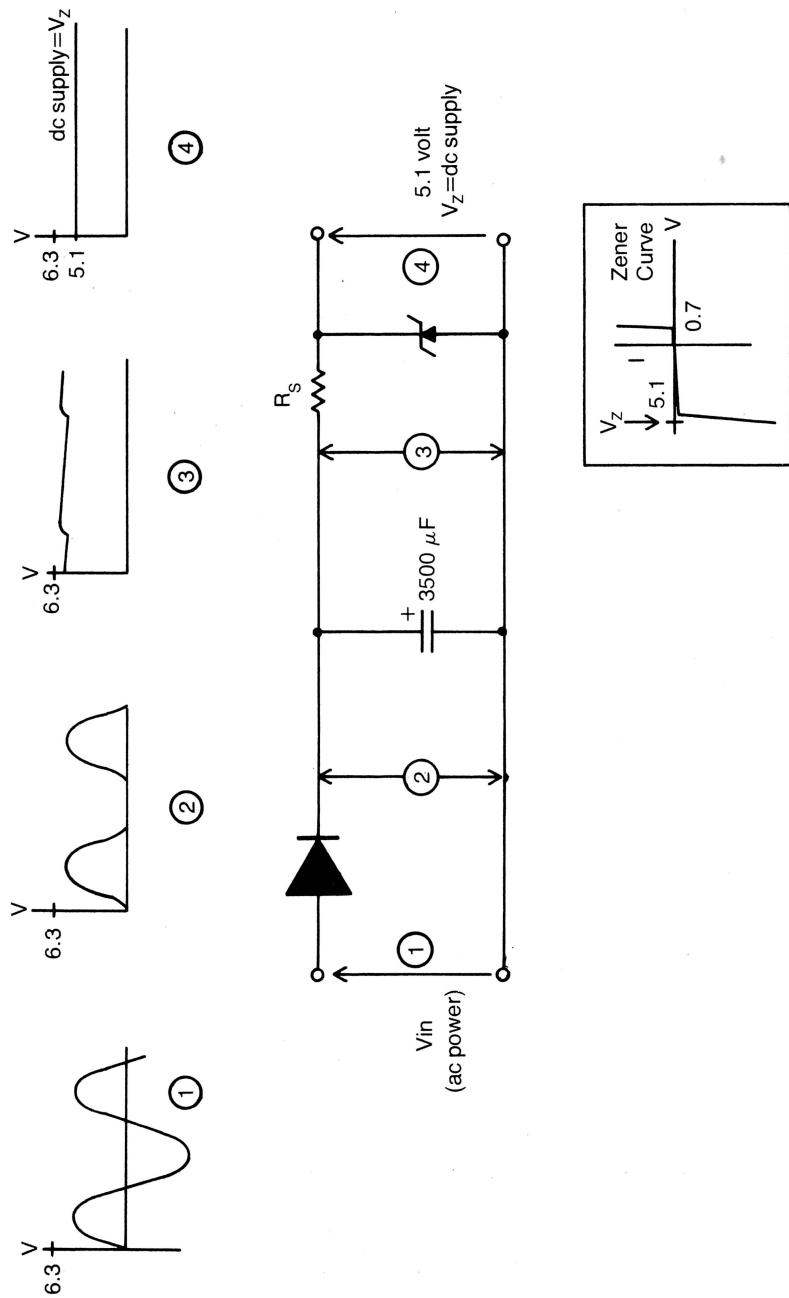


Fig. 6-10. Two diode applications are shown: as a power rectifier and as a zener regulator.

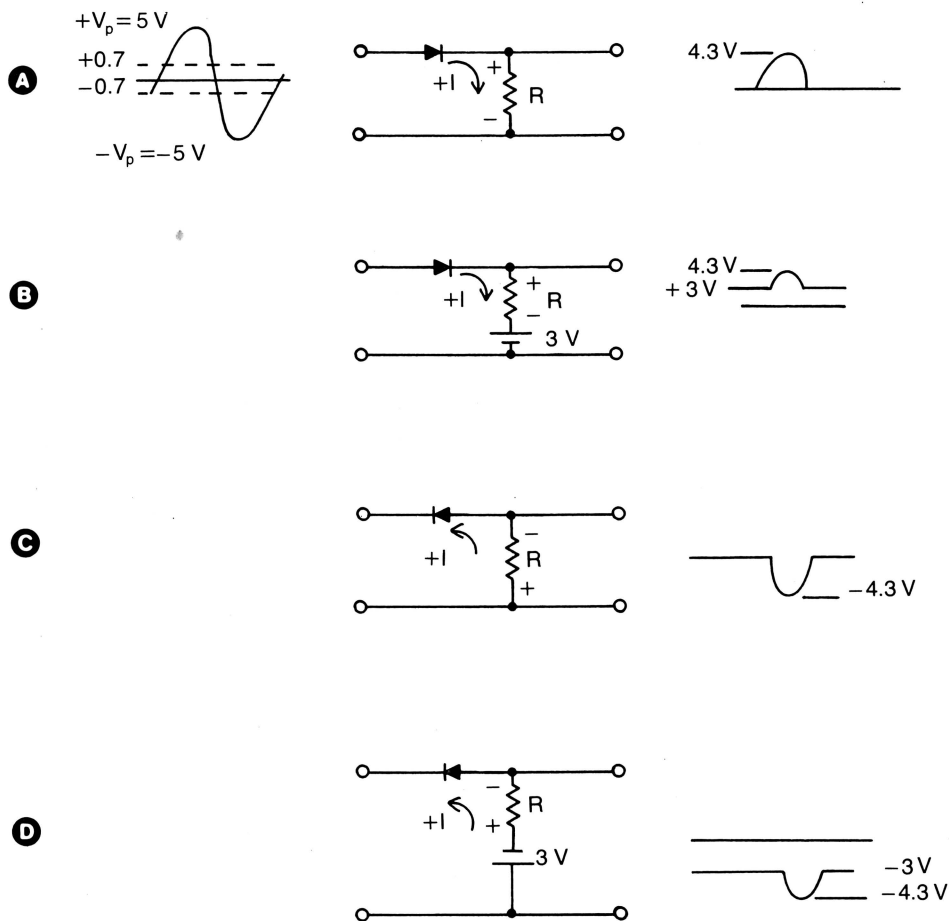


Fig. 6-11. Biased and unbiased series voltage clamp circuits. See text.

and other waveforms in a continuous fashion. This is useful in setting voltage levels for purposes of reference, for biasing other components, and for preventing excessive voltage and current from being applied to other parts of the circuit.

Diode Logic. The diode-resistor logic circuit in Fig. 6-13 is more of historical than practical interest, but it is instructive just the same. With both switches open, both diodes are forward-biased and conducting. Provided that resistors R_a and R_b are significantly less than the current-limiting resistor R_L , the output voltage will be low. In this case it will be equal to 0.9 V. When either switch A

or B is closed, one of the diodes will be conducting; V_{out} will be low, but this time it will have a value of about 1.1 V. It is only when both switches are closed (and both diodes are nonconducting) that V_{out} will be high or 5 V. This circuit therefore operates as an AND gate. OR, NOT, and other functions can be implemented using other configurations. Integrated logic (TTL/CMOS) has of course totally supplanted diode logic because of dramatic improvements in such factors as speed, power, component cost, power consumption, circuit size, and convenience.

LEDs. The p-n junction has been modified in

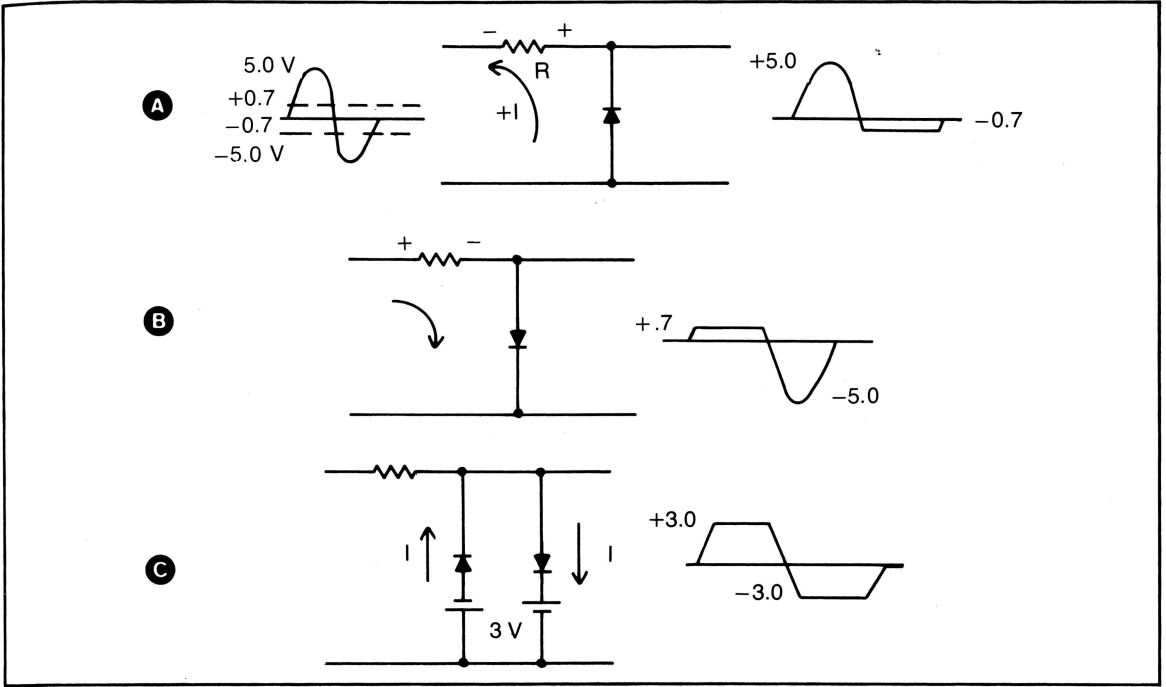


Fig. 6-12. A and B) Shunt voltage clamp circuits of clippers, biased and unbiased. C) Corer circuit.

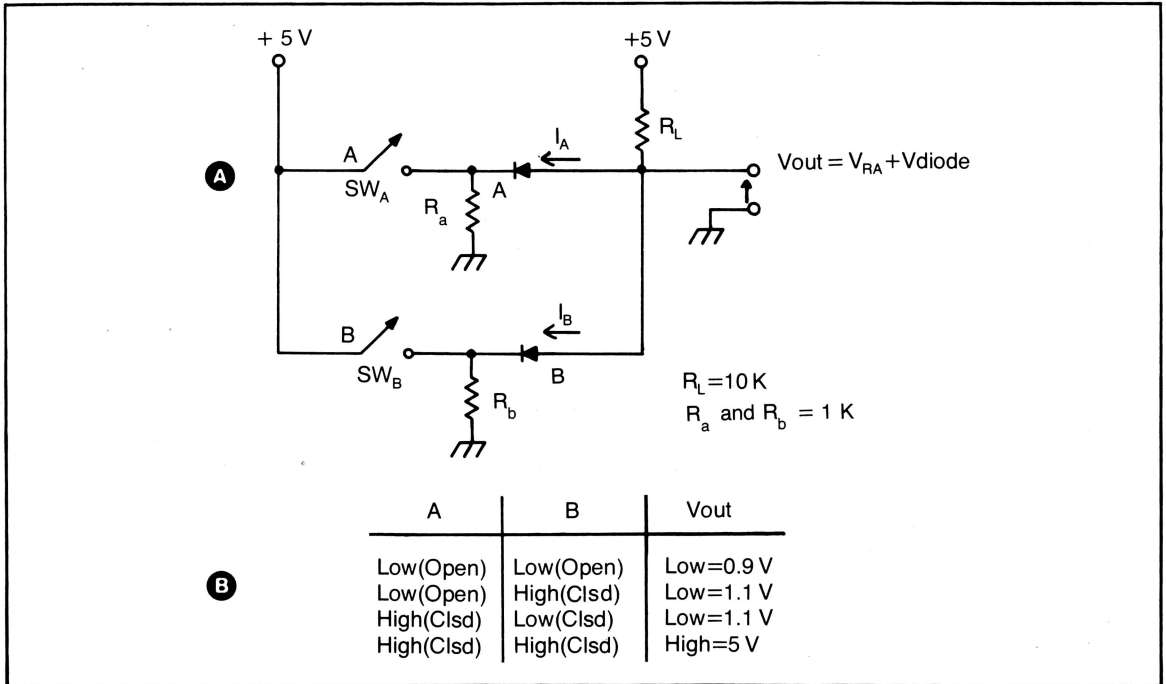


Fig. 6-13. A) Resistor-diode logic AND gate. B) Truth table.

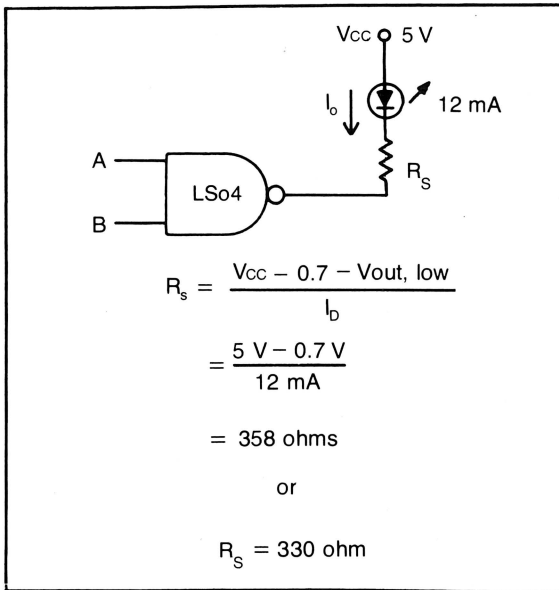


Fig. 6-14. The light emitting diode (LED). Calculation for the current limiting resistor, R_s is shown and is based on the nominal forward current rating for the diode.

the light emitting diode to give off visible and even infrared or laser light when properly biased. It is an extremely useful component as an indicator in communications and in on-board electrical isolation. The circuit shown in Fig. 6-14 shows an LED used to indicate the true state for a NAND gate. LEDs are created by the current required for them to light up. Note how the value of the current limiting resistor R_s is calculated, as given below the circuit. (12 mA exceeds LSTTL output ratings but might be acceptable in noncritical applications where only a simple indicator function is required.) A nominal value of 330 ohms would be satisfactory for R_s and is desirable as this is a commonly available value.

THE ESSENTIALS OF TRANSISTOR ACTION

There is nothing simple or easy about the physics of transistor operation or about the mathematical description of its behavior. However, you can gain a reasonably accurate idea of transistor action by thinking of this component as another black box. It has input and output parameters—voltage, current, and resistance—and a few simple

rules that relate these parameters.

Normally one thinks of a transistor as an amplifier or linear device, which it is. But in examining the transistor, we are concerned mainly with its operation as a switch that either conducts or does not conduct current. This is fortunate, because it is much easier to understand the transistor as a digital device which can assume either of two states than it is to understand it as a linear device which can assume any one of a near infinite number of states.

An explanation of the internal mechanisms that define transistor behavior is beyond the scope of this book. But you can understand many transistor properties merely by borrowing the concepts from the last section on the diode. The reason for this is that the transistor consists of two semiconductor diode junctions, one of which is forward-biased and the other reverse-biased. Most of the things mentioned in reference to the diode—very low resistance in the fully conducting state, high resistance in the nonconducting state, the existence of internal capacitance and its effects on waveforms (especially the square wave)—also apply to the transistor.

In other words, we are going to explore transistor behavior from the outside using as few parameters as possible, and also employ a simple model of its internal behavior. The emphasis will be on the *bipolar* transistor which is based on the pn junction.

Key Parameters

The pnp transistor consists of two pn junctions. The semi-schematic representation and the circuit schematic symbol are given in Fig. 6-15A. The *emitter* is so called because the majority carriers of the pnp transistor (holes) are injected into the device through this block of p-type material. This current is called the emitter current I_e , shown in Fig. 6-15B. The *base* is a thin wafer of n-type semiconductor through which the holes pass. In normal pnp operation, most of these holes (majority current) pass through the thin base into the *collector*; this current is known as the collector current, I_c . A very small hole current also flows in the base,

known as the base current, I_b . Relative to the n-type base, this tiny hole current is, of course, minority current.

One important relationship is that I_c is equal to the sum of I_c and I_b . This is nothing more than an application of Kirchhoff's Current Law! Because base current is so small, collector current is nearly equal to the emitter current, the difference being on the order of a percent or so.

We can talk about two currents then, a small base current and a much larger collector current. Collector current is effectively the amount of current flowing through the transistor. Further, it is the base current that determines the amount of collector current; I_c is a function of I_b . That is, the transistor is essentially a current dependent device. A small input base current causes a much larger output collector current to flow, hence the

transistor's ability to amplify current.

There is also a voltage associated with these changing currents that is called the collector-emitter voltage, V_{ce} . This can be considered as an output parameter, and represents the voltage drop across the collector and emitter terminals of the transistor. As it turns out, the more the transistor is forward-biased by increasing base current, the lower this collector-emitter voltage becomes.

In summary, the base current, I_b , is an *input* parameter and serves to *forward bias* the transistor into conduction. I_b determines the amount of collector current, I_c , and the collector-emitter voltage drop, V_{ce} . The latter two are *output* parameters. As I_b increases, I_c increases and V_{ce} decreases.

Turning out attention to the npn transistor you'll note that the only real difference is that the base is made of n-type semiconductor and the col-

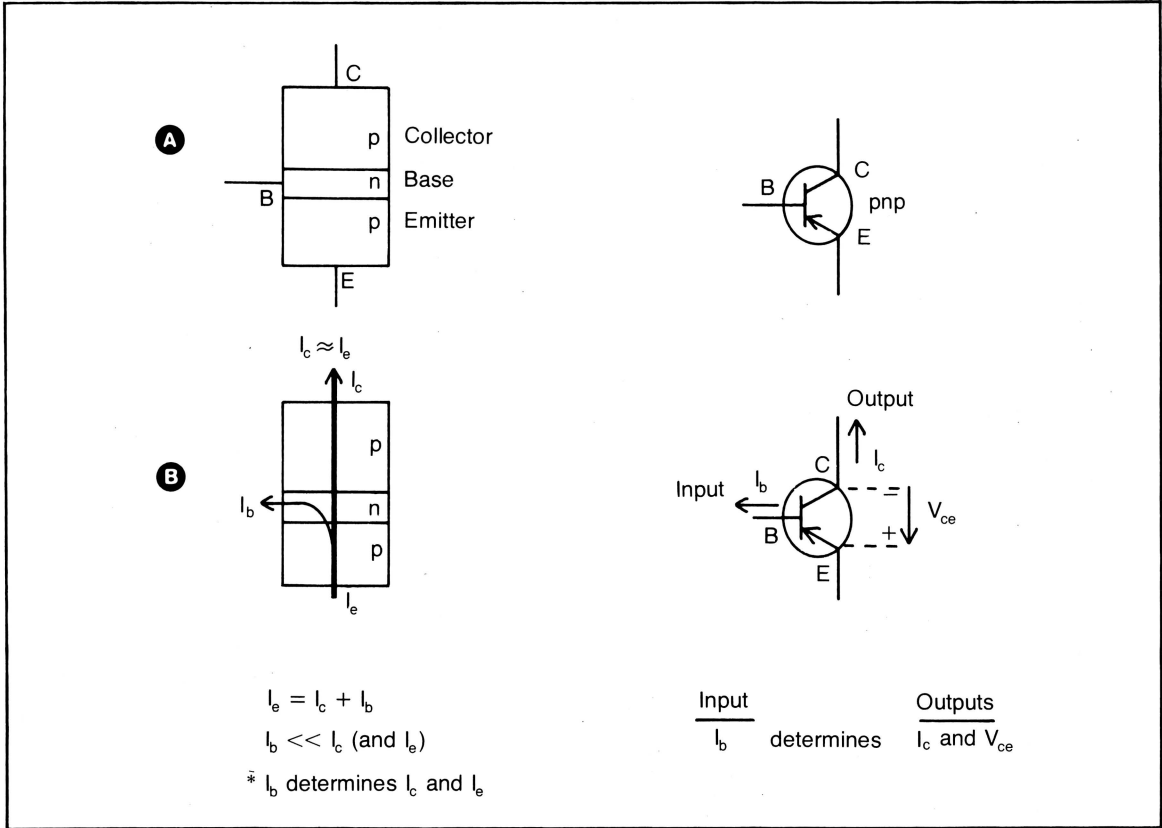


Fig. 6-15. A) Pnp semi-schematic and circuit symbol. B) Relationship of input and output parameters in the pnp transistor.

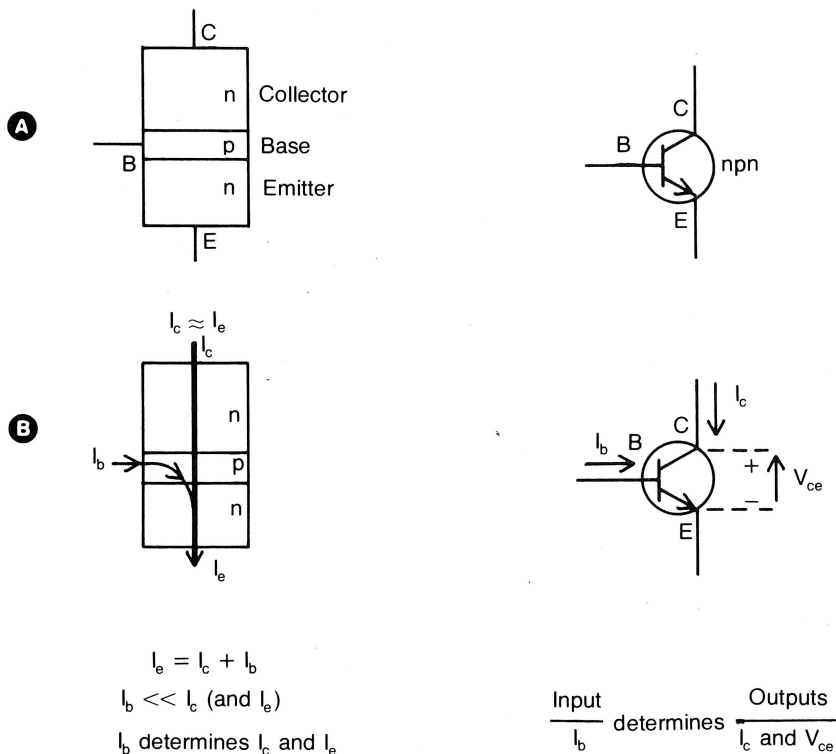


Fig. 6-16. A) Npn semi-schematic and circuit symbol. B) Relationship of input and output parameters in the npn transistor.

lector and emitter are made of p-type semiconductor. In the schematic circuit symbol in 6-16A, this is signified by the reversed direction of the arrowhead on the emitter lead. The majority carriers are electrons rather than holes. Therefore, you can draw the current flow arrows as before if you wanted to show this majority carrier (electron) flow. The direction of the current flow arrows would be the same—from emitter, through base, into the collector. However, I want to adhere to the positive current flow convention, and instead draw the current flow as shown in Fig. 6-16B.

Except for the reversed direction of positive current flow, everything said about the pnp transistor applies to this npn transistor. Collector and emitter currents are approximately equal, and collector current is taken to represent the effective

current flowing through the transistor. I_b is the major input parameter, and I_c and V_{ce} are the major output parameters. Again, I_b determines the degree of forward bias of the transistor, and hence the amount of collector current flowing through the transistor. As forward bias increases, I_{ce} increases and V_{ce} decreases.

The Transfer-Resistor

A convenient way of thinking about the transistor is that it functions like a variable resistor. Before we present this conceptual model, let's look a bit more closely at just how the majority carriers flow in a transistor. This will give you a bit more insight into the internal mechanism of transistor action and into the reasons for biasing the transistor

for normal operation. The npn transistor will serve as an example.

The circuit in Fig. 6-17 depicts an npn transistor in a *common-emitter* configuration. The emitter is grounded, the base serves as the input terminal and the collector serves as the output terminal. In this circuit, we will depart momentarily from positive current convention in order to see how the majority charge carriers, electrons, flow in the transistor.

The amount of current flowing in the base is signified by $-I_b$ and is determined by variable resistor R_b with V_{BB} being a fixed voltage source. Because this is an npn transistor, the polarity of this battery must be as shown in order for the base-emitter junction to be forward biased for normal operation.

The collector, on the other hand, is *reverse-biased*. For the npn transistor, this means that the collector voltage is positive, in this case +5 volts. This reverse bias is necessary for normal transistor action. The reasons for this can be explained as

follows: With a positive (reverse) bias on the collector the majority electrons from the emitter ($-I_e$) will be accelerated through the base to this strongly attractive positive potential. In fact, because the base region is relatively thin, most of the electrons will pass through the base into the collector and continue on as collector current, $-I_c$. The degree of this acceleration depends on the controlling influence of forward base-emitter bias, which is reflected by a small base current flow. Base-emitter bias could be increased by lowering R_b . This would have the effect of causing a large change in I_c for a relatively small change in I_b .

Note that if the collector were forward-biased with a negative voltage, this acceleration of electrons would not occur. Instead, all the electron flow would be from the collector and emitter into the base resulting in such a large base current that the transistor would be destroyed.

In short, the base-emitter junction is forward-biased and the collector-base junction is reverse-biased.

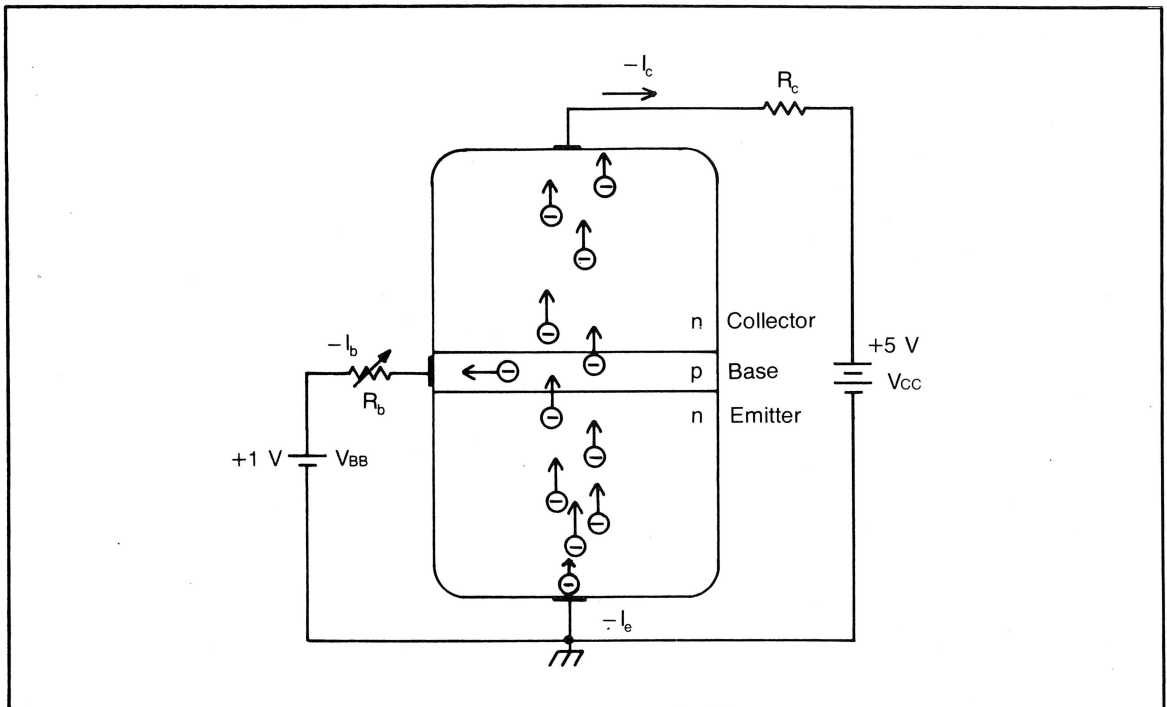


Fig. 6-17. Majority carrier (electron) flow in a forward-biased npn transistor.

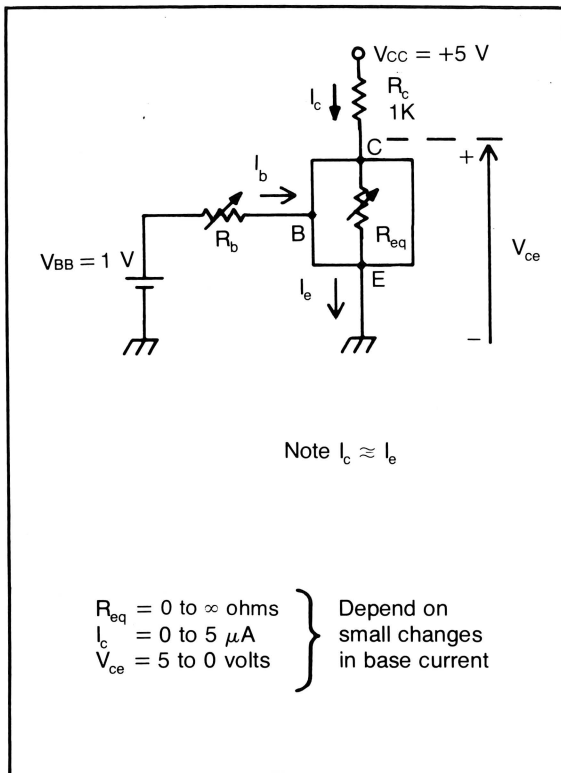


Fig. 6-18. Variable resistor model of transistor action. Analogous to a water valve. Small changes in base current cause large changes in collector current.

The idea of the transistor as a variable resistor is presented in Fig. 6-18. (Note that we are returning to the conventional positive current convention.) In truth, the name transistor is a contraction of transfer resistor and is meant to signify this variable effective resistance to current flow in the emitter-collector path.

Figure 6-18 is intended as a brief statement of transistor action. An increase in base current causes a decrease in effective transistor resistance. As a result more current flows for a given applied collector voltage. The transistor is more conductive because its effective resistance is less. (The fancy name for this is *transconductance*, a term used in the more formal treatments found in engineering texts.) Naturally, the voltage drop across this resistance, R_{eq} in the figure, will decrease as its value decreases. More of the collector supply voltage,

V_{CC} , will drop across the fixed collector resistor, R_c . This is a result of simple voltage division across series resistors.

This can be summarized by saying that the output parameters, V_{ce} and I_c , change just as if base current had the effect of changing the value of a variable resistor. In the ideal case, a nonconducting transistor, one in the OFF state, would have:

- ☐ An infinite resistance.
- ☐ Zero current flowing through it.
- ☐ A voltage drop V_{ce} equal to the supply voltage, + volts in this example.

In the fully conducting or on state the transistor current is at maximum, the exact value being limited by the resistor R_c . In this so-called *saturated* state the transistor would have:

- ☐ An effective resistance of zero ohms.
- ☐ A current determined by V_{CC}/R_c , or 5 mA in this example.
- ☐ A voltage drop V_{ce} of zero volts.

When the transistor is on we say that it is saturated. It is saturated in the sense that the current through the collector-emitter circuit is at a maximum, because the transistor resistance is at a minimum—near zero, practically speaking.

Voltage and Current Transfer Curves

Knowing the shapes of the I_b versus I_c curve and I_b versus V_{ce} curve helps to understand the nonconducting and saturation states that are important in the transistor's digital applications. We call these curves *transfer curves* because they show how an input parameter (current, I_b) is translated into an output parameter (I_c or V_{ce}).

To explain these curves the circuit in Fig. 6-19 is used; it shows the common or grounded-emitter configuration of an npn transistor. The source and base bias voltages are +5 and +1 volts, respectively. The combination of the 0 to 500 K potentiometer (R1) and the fixed 2 K resistor (R2) in the base circuit allows you to vary the base current from 2 μA (effectively zero current) to about 500

μA . Current limiting resistor R_1 in the collector circuit is 1 K. With only this information, plus the principles already introduced, we can say a number of things about the operation of this circuit.

First, the relationship of emitter, base, and collector currents is restated in the figure. Again, the emitter and collector currents are nearly equal because I_b is so small. There are also corresponding inter-lead voltages that should be mentioned. Using Kirchhoff's Voltage Law, you can see that the collector-emitter voltage is equal to the sum of the base-emitter and collector-base potentials. The polarities of these voltages are indicated by the signed arrows in the figure. Note that the base-emitter junction voltage, V_{be} , will be limited to a value of 0.7 volts, even when the transistor is fully forward-biased. This is just like the diode. There-

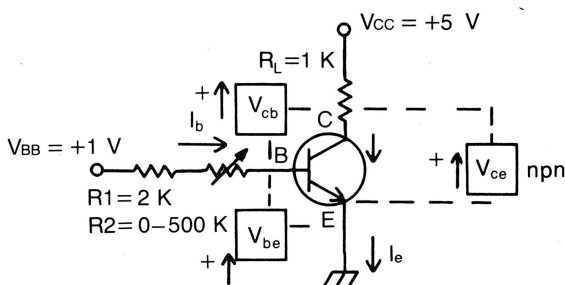
fore, $V_{ce} = V_{cb} + 0.7$ for the forward-biased state.

The transistor input, the base current, is estimated by taking the voltage drop across both base resistors and dividing by their total series value. The junction potential of 0.7 volts is subtracted from the base bias voltage in order to get this voltage drop, as given in the Fig. 6-19:

$$I_b = (V_{BB} - 0.7) / (R_1 + R_2)$$

A value of a few μA represents an effective zero input current, while that of a few hundred μA or more will yield a saturation state.

(The amount of base current needed to drive the transistor into a saturation state depends on the transistor's current-amplifying ability and on the load resistance in the collector circuit, as you'll see shortly.)



Currents	$I_e = I_c + I_b$	\longrightarrow	$I_e \approx I_c$
Voltages	$V_{ce} = V_{cb} + V_{be}$	\longrightarrow	$V_{CE} = V_{cb} + 0.7$

Inputs $I_b = (V_{BB} - 0.7) / R_1 + R_2$

Outputs

$$\begin{aligned}
 V_{ce, \text{off}} &\cong V_{CC} = +5 \\
 V_{ce, \text{sat}} &\cong 0\text{V} \\
 I_{c, \text{off}} &\cong 0\text{A} \\
 I_{c, \text{sat}} &\cong \frac{V_{CC} - V_{ce, \text{sat}}}{R_L} \cong 5\text{mA}
 \end{aligned}$$

Fig. 6-19. On (saturation) and off states of a transistor. See text.

The outputs for on and off states of the transistor are given below in the figure. The values given are close approximations. When there are only a few μA flowing in the base circuit, the transistor is essentially off since there is insufficient current for any significant conduction to take place. In the off state zero current flows and the voltage drop across the near infinite collector-emitter resistance will be about equal to the supply voltage V_{CC} .

Alternatively, when I_b is high enough, the transistor is saturated, and I_c will be the maximum permitted for the given value of R_L . Collector current in this on or saturated state will be equal to the voltage drop across the load resistor R_L divided by its value:

$$I_{c, \text{sat}} = (V_{CC} - V_{ce, \text{sat}}) / R_L$$

When saturated, the effective resistance of the transistor will by definition be near zero. That is, $V_{ce} = 0$ for all practical purposes. Therefore, the current through the transistor is approximated by V_{CC}/R_L or about 5 mA.

Of course, in between the off state and the on (saturated or zero resistance) state there is a broad intermediate range of values for collector current and for collector-emitter voltage. I_c and V_{ce} will change continuously in response to changes in the input current I_b . They will, in fact, mimic the input waveform. The collector current will be an amplified version of the input base current, and depending on the exact values of the base and collector resistances, input voltages variations will also be amplified. Because the output parameters change in a proportional manner, the amplification is said to be linear. This linear of transistor function is the concern of those using transistors for such applications as conventional voice and picture transmission and reception, instrumentation, measurement, and other areas where an amplified signal faithful to the original is required.

Given the *end points* of collector current and voltage—their values in the on and off states—we can construct the transfer curves by simply drawing a straight line between them. A simple common-emitter circuit is shown in Fig. 6-20A, and its I_c and

V_{ce} are plotted against I_b in Fig. 6-20 B and C.

Figure 6-20B is the graphic representation of the relationship between collector current and base current. Between the end points—zero collector current (off) and maximum collector current (on)—there is a linear zone in which collector current increases in direct proportion to increases in base current. Observe that microamp changes in I_b result in milliamp changes in I_c . In a very literal sense, then, this transfer curve for I_c reflects the *amplification* of base current. This amplification factor is the ratio of collector to base current and is called the transistor's *beta*, or *current gain*. The current gain or beta value of a typical general purpose transistor will usually fall in the range of from 20 to 200. In the plot of 6-20B, the *slope* of the curve represents the value of beta. For the sake of illustration, we'll assume that this transistor's beta value is 50.

Figure 6-20C is the I_b versus V_{ce} transfer curve. It is another graphic aid to illustrate that the voltage drop across the transistor drops linearly from the supply voltage towards zero, as I_b (forward bias) increases. Again, note the off-linear-on configuration on this voltage transfer curve.

An important point about the inter-lead voltages at saturation is this: The collector-base potential is positive when the transistor is unsaturated, but becomes negative when it is at or very near saturation. This can be proven from the voltage equation that is restated in Fig. 6-20A:

$$V_{ce} = V_{cb} + V_{be}$$

The base-emitter (pn) junction remains at about 0.7 V, so in the forward-biased state we can state the following:

$$V_{ce} = V_{cb} + 0.7$$

At saturation, the collector-emitter voltage goes to near zero volts because the effective transistor resistance is negligible, therefore,

$$0 = V_{cb} + 0.7$$

or

$$V_{cb} = -0.7 \text{ volts}$$

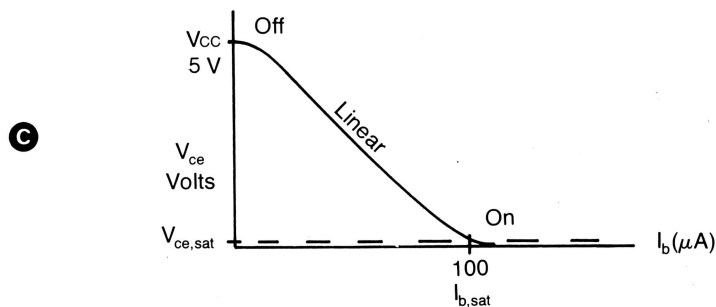
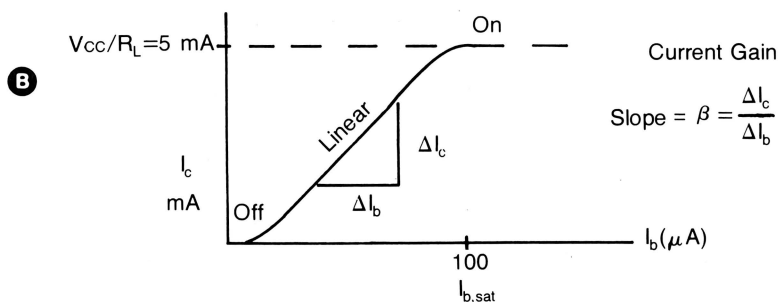
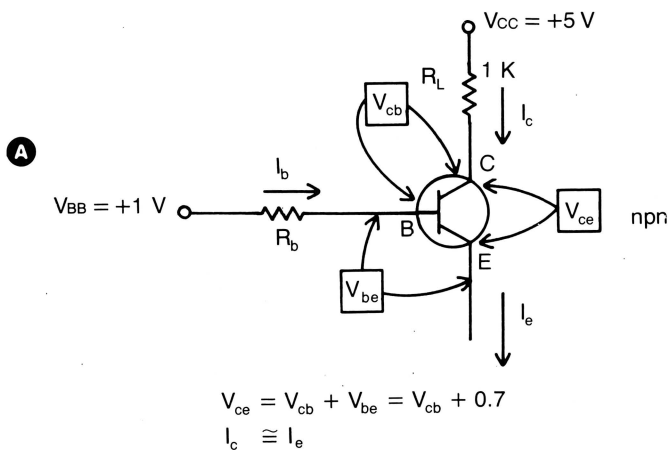


Fig. 6-20. Transfer curves. A) Currents and voltages in the test circuit. B) The slope of the current transfer curve represents the transistor current gain. C) The voltage transfer curve illustrates how V_{ce} drops off linearly as forward bias (base current) is increased.

V_{cb} is also negative just short of saturation. The importance of this is two-fold. First, you can test any transistor for saturation while it is operating in a circuit. Merely place a voltmeter across the collector-base leads. If the voltage is negative, the transistor is at or very near saturation.

More significant is that when V_{cb} is very slightly negative, say -0.2 to -0.3 volts, the transistor is also near saturation. In fact, if you could in some way hold V_{cb} slightly positive, you would have near maximum collector current and near zero collector-emitter voltage. Full saturation entails an excess of mobile charge carriers, and so the time it takes for the transistor to turn off is relatively longer than when it is not saturated. Therefore, the advantage of keeping the transistor just short of saturation is that switching time is greatly improved.

One way of improving switching time, specifically the lag from the on to off state, is to use the *Schottky diode*. As illustrated in Fig. 6-21A, there is an expected rising and falling time for the transistor output voltage (exaggerated here) in response to a square wave input. Because of the greater number of majority carriers in the conducting phase, it takes longer for the device to turn off than to turn on. In this instance t_{LH} is greater than t_{HL} . This is just as you would expect from any semiconductor device.

The Schottky diode, Fig. 6-21B, is specially designed to have a low forward bias of about 0.3 V. This means that the V_{cb} is limited -0.3 volts, thereby keeping the transistor just short of complete saturation. Any excess base current is fed into the collector through the Schottky diode instead of into the base-emitter junctions. This avoids excess charge carriers, and so lessens the discharge time,

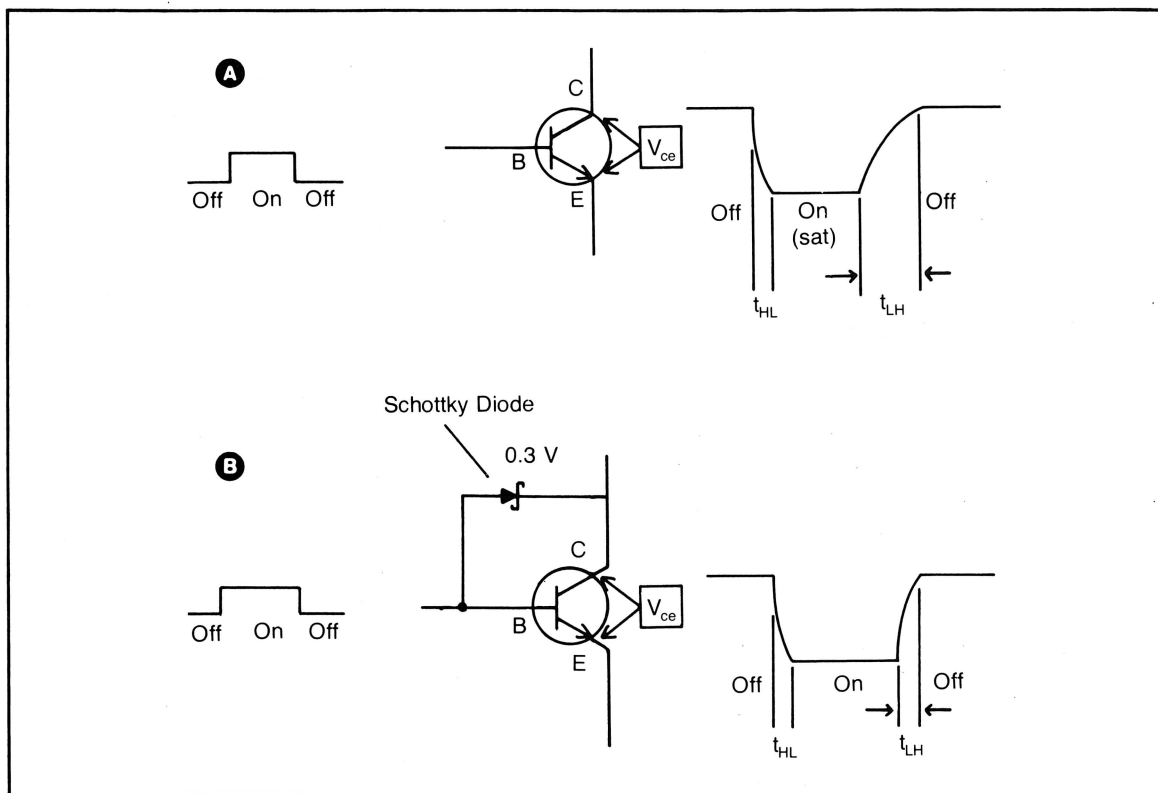
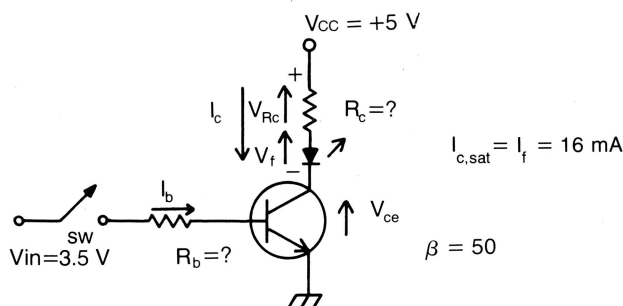


Fig. 6-21. Use of the Schottky diode to improve switching time. A) Standard transistor. B) Transistor with Schottky diode clamps V_{cb} to -0.3 V, preventing total saturation. This avoids excess charge carriers in the base-emitter junction and improves switching speed (esp. on to off transition time).



Given: V_{in} , $I_{c,sat}$, V_{CC} , β

Find: R_b , R_c

$$R_c = \frac{V_{RC}}{I_{c,sat}} = \frac{V_{CC} - V_{ce} - V_f}{I_{c,sat}}$$

$$I_b \geq \frac{I_{c,sat}}{\beta}$$

$$R_b \leq \frac{V_{in} - V_{be}}{I_b}$$

Fig. 6-22. Common-emitter with series load (current sink). See text for discussion.

t_{LH} , and improves overall speed. This is the basis for Schottky logic in LSTTL.

USING THE TRANSISTOR AS A SWITCH

As stated, it is not the linear but rather the on and off states of conduction we are concerned with in digital applications. Current gain is necessary just the same, even in digital circuits. Having a beta of moderate value allows you to drive other circuitry: it is useful to be able to switch relatively large (mA) values of current on and off using only a small change in input or base current (a hundred μA or so). Switching moderate currents on and off allows one to control other on/off devices, such as lamps, relays, speakers, and digital ICs. But since we don't care about faithfully duplicating waveforms or about the exact shape of the linear zone (as we would in most analog applications) our use of the transistor is greatly simplified. This aspect of the transistor as a digital or two-state device is the subject of this section.

Current Sink or Series Configuration

As a switch, the transistor can drive resistive loads in either of two configurations: the *sink* or *source* configurations, which are also known as the *series* and *shunt* configurations, respectively. In using the transistor as a switch in either of these configurations, our main concern is how to drive it into saturation, namely, to determine just what base current is necessary to cause V_{ce} to fall to zero and the maximum I_c to flow.

Using the common-emitter arrangement in Fig. 6-22, we will determine the value of R_b needed to provide saturation for the npn transistor shown. In the circuit, we are driving an LED which is the *load* or current-consuming component at the transistor's output. This LED is in series with the transistor. Further, the current going through this series element is sinking to ground. The collector, which is the circuit's output terminal, is said to be in a *current sink* configuration with respect to the load, because it is sinking the load current to ground.

Alternatively, we can say that **a load is in current sink configuration when it is connected between the output terminal and the supply voltage.**

An input base voltage of +3.5 volts is to be switched on or off in order to supply the driving current for the LED. Assume that the nominal forward current (I_f) needed for satisfactory brightness is 16 mA, a value one might obtain from a hypothetical data sheet for the LED. This is the current we want flowing in the collector circuit when the transistor is on or saturated.

Although it may seem to be putting the cart before the horse, it is necessary to calculate the value of the current limiting resistor in the collector circuit first. We assume that when the desired current of 16 mA is flowing that the collector-emitter voltage, V_{ce} , will be zero. $I_{c,sat}$ is 16 mA. The source voltage, V_{CC} , will be zero. $I_{c,sat}$ is 16 mA. The source voltage, V_{CC} , is +5 volts. The voltage drops, by Kirchoff's Voltage Law, will be the sum of the voltage drops in the collector-emitter circuit: the voltage drop across R_c (V_{RC}), across the LED (V_f) and across the transistor (V_{ce}):

$$\begin{aligned} V_{CC} &= V_{Rc} + V_f + V_{ce} \\ \text{OR} \\ V_{Rc} &= V_{CC} - V_f - V_{ce} \end{aligned}$$

We can calculate R_c by taking the voltage across the resistor divided by the saturation current flowing through it, using the above equation for V_{RC} .

$$\begin{aligned} R_c &= V_{Rc} / I_{c,sat} \\ &= V_{CC} - V_f - V_{ce} / I_{c,sat} \\ &= (5.0 - 0.7 - 0) \text{ volts} / 16 \text{ mA} \\ &= 268 \text{ ohms.} \end{aligned}$$

This value of 268 ohms is not a common value. We choose the nearest most common higher value for R_c , 330 ohms. The reason for choosing the higher value has to do with assuring that V_{ce} is zero. We may want to use the V_{ce} as a TTL logic low in a digital application. If you used a lower value for R_c , more collector current would flow. Because of the nature of transistor action, the transistor would come out of saturation: the excessive current flow

would cause a voltage drop across the collector emitter circuit. As a result you would have a positive, nonzero, logic low which is not a desirable state of affairs.

Next, to calculate the value of R_b , we need to know the saturation collector current (which we still assume is 16 mA) and the current gain of this transistor, which we'll say has a nominal value of 50. From the relation $I_c = \beta \times I_b$ we can write:

$$\begin{aligned} I_b &= I_c / \beta \\ &= 16 \text{ mA} / 50 \\ &= 0.32 \text{ mA or } 320 \mu\text{A.} \end{aligned}$$

Then the value of the base resistor needed to allow this current to flow will be the voltage drop across it divided by this current:

$$\begin{aligned} R_b &= (V_{in} - V_{be}) / I_b \\ &= (3.5 - .7) \text{ volts} / 320 \times 10^{-6} \\ &= 8750 \text{ ohms.} \end{aligned}$$

This value of 320 μA is the minimum value of current needed to cause 16 mA to flow in the collector, and therefore 8750 ohms is the maximum value of the base resistor allowed. Most transistor beta ratings are only approximate, and so it is common practice to assume that the current gain might be much lower for any given transistor due to variations in its manufacture. In fact, actual beta could be $\frac{1}{3}$ to $\frac{1}{2}$ of the nominal value! A reasonable course is to use a base resistor value around $\frac{1}{3}$ the value of the base resistor to assure sufficient base current, and then to take the nearest commonly available value for R_b . A 3300 ohm resistor would be satisfactory to provide sufficient base current for a saturation collector current of 16 mA.

You may be wondering about the situation in which beta is not low, but actually higher than calculated, again due to variations in fabrication. Perhaps the current gain is 50, or even 100. Doesn't the base current then serve to drive the collector current higher than 16 mA? The answer is no, because the transistor is saturated. V_{ce} already is zero, so current in the collector circuit cannot go higher. The reason for this is that the current

limiting resistor, R_c , prevents this. What happens to the excess I_b then? It flows through the base-emitter junction to ground. This results in excess charge carriers in the base-emitter circuit, but provided that I_b does not exceed a maximum rating (usually well into the milliamp range) there is no harm done to the transistor. However, the excess charge carriers do cause problems with the transistor's switching speed, already mentioned. Hence the use of Schottky diodes to avoid complete saturation.

The common-emitter circuit with appropriate resistor values is repeated in Fig. 6-23A. The point of this illustration is to show the shape of the output voltage, V_{ce} , given a 3.5 volt square wave as the input. When $V_{in} = 0$, the transistor is off, and has infinite resistance. Therefore, all the source voltage drops across it, and because no current flows,

the LED is off. When V_{in} goes to 3.5 volts, the transistor is saturated and V_{ce} becomes zero volts. Saturation current flows and the LED lights up. Note that the shape of the output voltage is the inverse of the input signal. The transistor in a common-emitter configuration acts as a voltage inverter if we use V_{ce} as the output voltage.

The circuit in Fig. 6-23B, on the other hand, is an emitter-follower. The voltage is taken across an emitter resistor, R_e , which is also 330 ohms. The transistor saturates as before, with the same value of $I_{c,sat}$. However, we are taking the output voltage across the emitter circuit (the resistor R_e) rather than across the transistor itself. When the transistor is on, the saturation current causes a voltage drop across R_e . Therefore, in this emitter-follower configuration the output voltage, V_e is the noninverted form of the input waveform. When V_{in} is 3.5

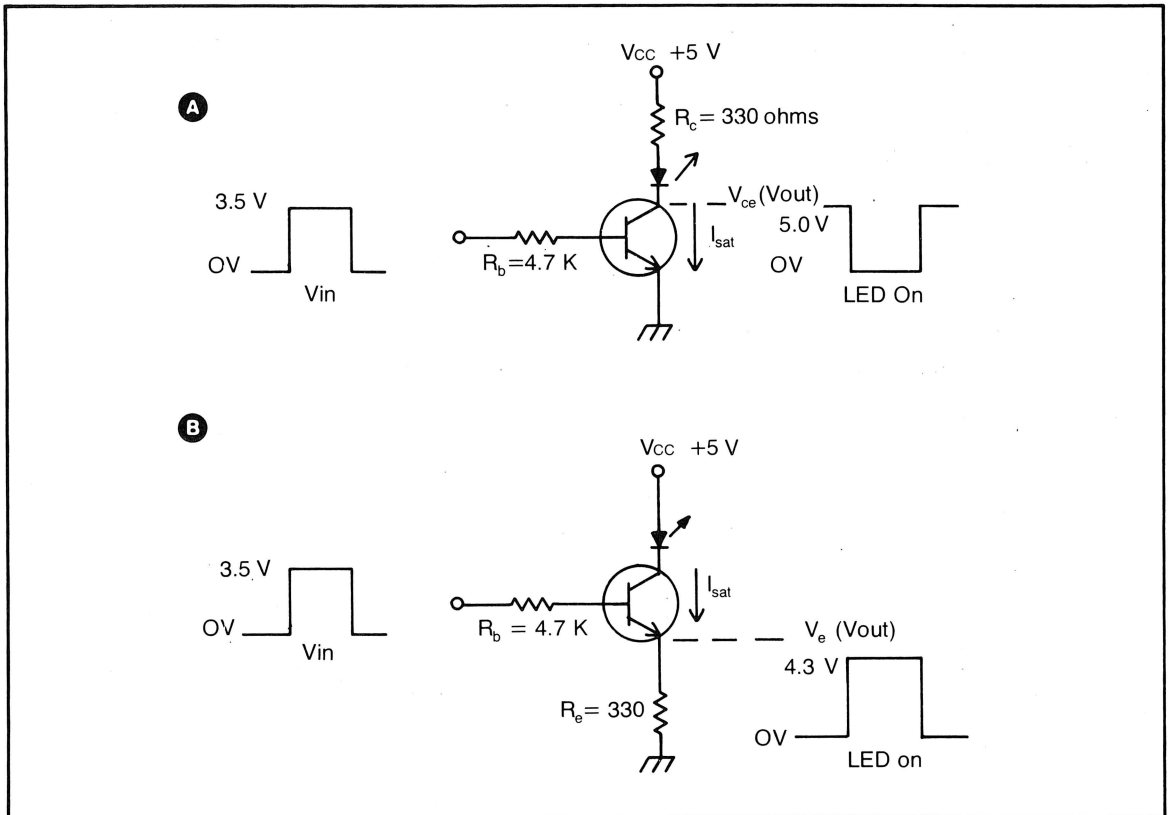


Fig. 6-23. A) Common-emitter with series load acting as an inverting switch. B) The emitter-follower acting as a noninverting switch.

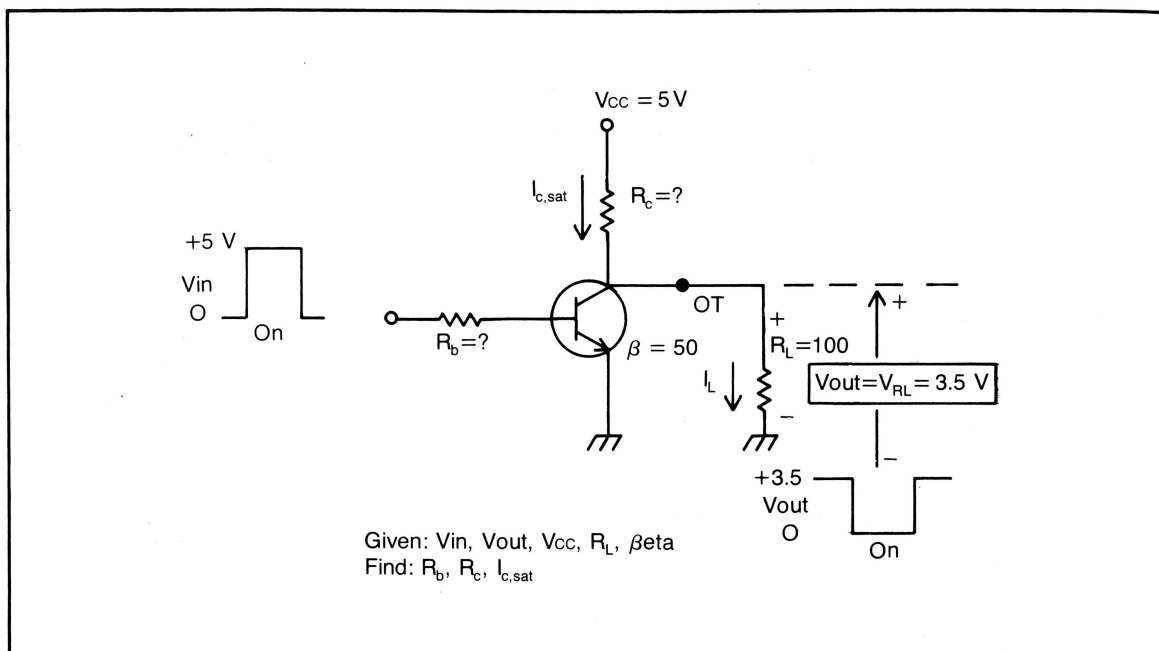


Fig. 6-24. Example problem of a common-emitter with shunt load (current source). Solution given in Fig. 6-25.

V , the output is 4.3 V, and the LED lights up. When V_{in} goes to zero, I_c is also zero, so V_e is zero and the LED is off.

By the way, what is I_c and V_{out} for this emitter-follower? First, $I_{c,sat}$ is 13 mA— $(5.0 - 0.7)/330 = 13$. Therefore, the amplitude of V_e (or V_{out}) will be $330 \text{ ohms} \times .013 \text{ amp}$ or 4.3 volts. This is what you would expect if you assumed V_{ce} were zero and then simply subtracted the diode forward drop of .7 volts from the 5 volt source.

Note that in the inverting (common-emitter) and noninverting (emitter-follower) forms of this series load (current sink) configuration, the voltage levels are in the TTL range. That is, both V_{in} and V_{out} assume TTL-compatible logic high and low values. In fact, that is the whole point of what preceded, namely that TTL devices may be connected to the input or output of such a transistor. This is occasionally necessary if you want to boost the output power of a low power digital IC (e.g., LSTTL) to drive a high current device (relay, lamp, transmission line) or to drive very many other digital ICs with the same inverted or noninverted signal.

Current Source or Shunt Configuration

In the common-emitter circuit of Fig. 6-24, we again want to turn the npn transistor either off (nonconducting) or on (saturated). This time, however, the resistive load R_L is shunted across the transistor, from the collector output terminal, OT, to ground. From the viewpoint of the load, this output terminal is a source of current. Hence the name, current source or shunt configuration. Generally, we can say that **in current source arrangements, the load is connected between the output and ground.**

There is an important difference in the way we define the function of sink and source configurations. In the current *sink* of the previous section the main determining parameter was how much *current* we wanted through the load. In the current *source*, however, it is the *voltage* across it that is the key variable. While this distinction is not a hard and fast one for all applications, it is nonetheless a very useful one for understanding the calculations involved.

In Fig. 6-24, you can see that the load resistance (whatever its identity may be) is 100 ohms,

and that this resistive element must have +3.5 volts across it if it is to function properly. It will be switched on and off by a +5 volt square wave. The transistor has a current gain of 50, and the VCC equals +5 volts. We must find the values of R_c , R_b , and $I_{c,sat}$. Note that the output voltage waveform is again inverted with respect to the input waveform. Do you know why?

The solution to this circuit is in two parts, just as before.

In the first part, we again deal with the output or collector side of the circuit. With the transistor off or nonconducting, all the current flows through the R_c to R_L branch, because the transistor is a virtual open circuit. Simply stated, this part involves nothing more than voltage division: we must find the value of R_c which provides for 3.5 volts across R_L . As given in Fig. 6-25A, the equation is:

$$\begin{aligned} R_c &= (V_{CC}/V_{out} - 1)R_L \\ &= (5/3.5 - 1) \times 100 \text{ ohms} \\ &= 42.8 \text{ ohms} \end{aligned}$$

Double check this by:

$$\begin{aligned} V_{out} &= (R_L/R_L + R_c) V_{CC} \\ &= 100/142.8 \times 5 \text{ V} \\ &= 3.5 \text{ volts} \end{aligned}$$

Naturally, 42.8 ohms is not a readily available value. However, we can approximate it very closely by connecting a 47 ohm and 470 ohm resistor in parallel:

$$\begin{aligned} R_{eq} &= (47)(470) / (47 + 470) \\ &= 42.7 \text{ ohms} \end{aligned}$$

Important note: the load current I_L —the current which happens to flow through the load resistance—is not a determining variable in this particular problem. It simply follows from the specified value of load resistance and the load voltage required. Hence, like it or not, I_L is 35 mA. Alternatively, we could have stated the problem differently: If we wanted to specify load current instead, then the voltage drop across the load would have to fall as it may from $V = IR$.

The second part of the solution is given in Fig. 6-25B. In this part, the transistor is on or saturated.

We want zero voltage across the load in this situation. Also, there will be no current flowing through the load because the transistor will have near zero resistance. Either way you state it, V_{out} will be near zero when V_{in} is high, or +5 volts. How then do we assure saturation?

First, the value of R_c followed from the output voltage requirement. From this, and the fact that $V_{ce,sat}$ is zero in saturation, we can calculate $I_{c,sat}$:

$$\begin{aligned} I_{c,sat} &= (V_{CC} - V_{ce,sat}) / R_c \\ &= V_{CC}/R_c \\ &= 5 \text{ volts} / 42.8 \text{ ohms} \\ &= 117 \text{ mA} \end{aligned}$$

Then the minimum base current needed to provide this current is:

$$\begin{aligned} I_b &= I_{c,sat} / \beta \\ &= 117 \text{ mA} / 50 \\ &= 2.34 \text{ mA} \end{aligned}$$

And, the base resistor is:

$$\begin{aligned} R_b &= (V_{in} - V_{be}) / I_b \\ &= (5 - 0.7) \text{ volts} / 2.34 \text{ mA} \\ &= 1838 \text{ ohms} \end{aligned}$$

In order to assure saturation, a value of about $\frac{1}{3}$ the calculated value could be used. Some would recommend the use of a 500 ohm resistor. This is necessary because beta values can be much less than that suggested in a data sheet, especially for those transistors obtained from the surplus market. Again, the rule is to assume the worst case for beta: better to oversaturate with too much base drive current, than to undersaturate and fall short of the desired V_{ce} of near zero. And you assure such saturation by using $\frac{1}{2}$, and preferably $\frac{1}{3}$ or even $\frac{1}{4}$ the value of R_b calculated from the nominal beta.

As to why this shunt or current source configuration has a V_{out} that is the inverse of V_{in} , we can use either of two lines of reasoning. The more obvious reason is that when V_{in} is high, the transistor is saturated and therefore the voltage (V_{ce}) across R_c is near zero. Alternatively, you can say that when the transistor is on, all the current is shunted through it rather than through the load

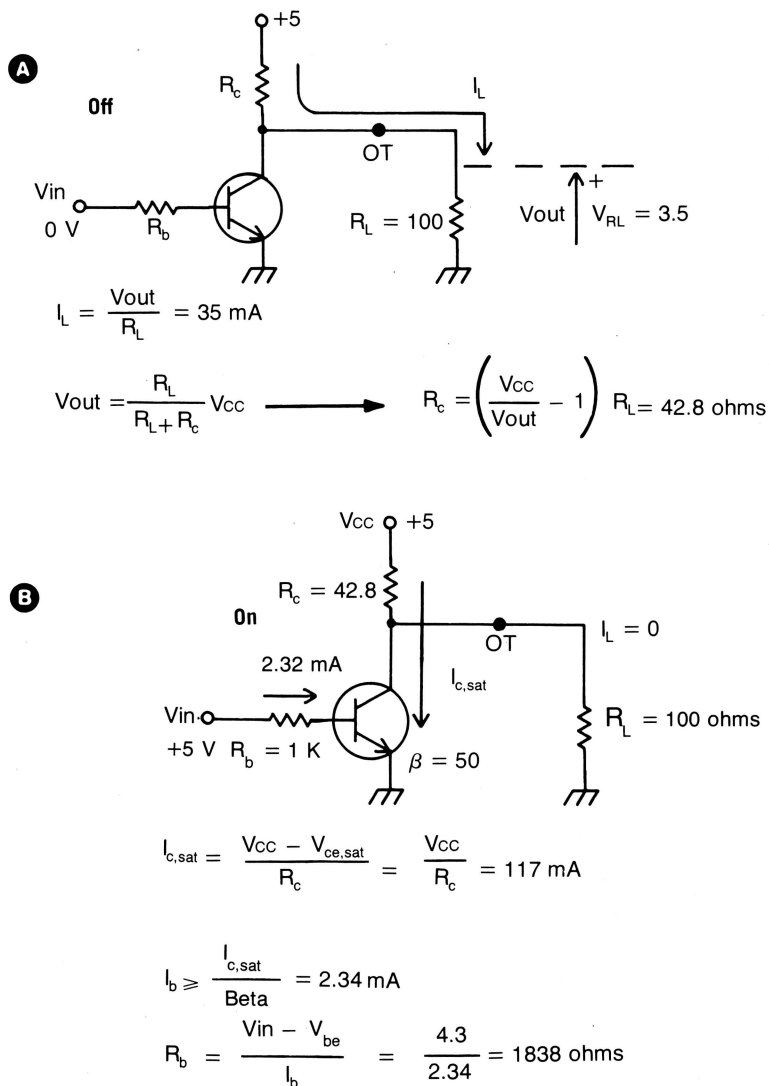


Fig. 6-25. A) Solving for R_c involves simple use of the voltage division principle. Note that I_c is 35 mA. See text. B) Solving for I_b involves finding $I_{c,sat}$ from this value of R_c , then proceeding from the current gain relationship.

resistance. Hence, with no load current flowing, there will be no voltage drop across that resistance.

Overloading and Desaturation

When using the transistor in digital applications it is good policy to adhere to the prescribed logic levels for the family of digital IC employed.

TTL output logic levels are specified as being no less than 2.4 volts for an output high and no more than 0.4 volts for an output low. The question arises as to just what factors can cause a transistor's output to fall short of these requirements.

The answer is simple: excessive *loading*. Specifically, excessive loading means that the

load—be it a discrete component or some other module in the circuit—draws too much current. We can phrase this another way by saying that if the load resistance itself is too low, then it draws more current than is desirable. The implications of excessive loading are different for the two types of circuits already discussed.

In the current sink configuration, a series of resistive loads are connected between the collector terminal and a common +5 volts supply, V_{CC} . As you can see from Fig. 6-26A, the loads R_1 , R_2 , and R_3 are in series with the transistor. However, they are in parallel with one another.

First, assume that the transistor is saturated, with only R_c and R_1 in place. V_{out} , which is the same as V_{ce} , will be zero. But if more and more loads are added— R_2 and R_3 —the equivalent resistance obviously decreases. (Remember that the R_{eq} of parallel resistances is less than the smallest of them.) The effect of decreased load resistance is

that more current will flow when the transistor is on. A property of the transistor is that if it is saturated, and you then cause more total current to flow through the collector-emitter circuit (I_T), it will come out of saturation! That is, the excess load current causes a voltage drop across the collector/emitter element, and therefore the output low state rises.

The effect of increasing load current (decreasing load resistance) is shown in Fig. 6-26B. This circuit operates with a near zero logic low provided R_1 is the only load in place. With added loads, the voltage low value becomes marginal, and finally rises above the acceptable minimum for a TTL output low. In short, overloading a current sink causes a rise in the logic low level.

Obviously, when using discrete transistors in digital circuits, you must assure that the transistor is fully saturated when turned on (conducting) in order to assure a valid, near zero, logic low level.

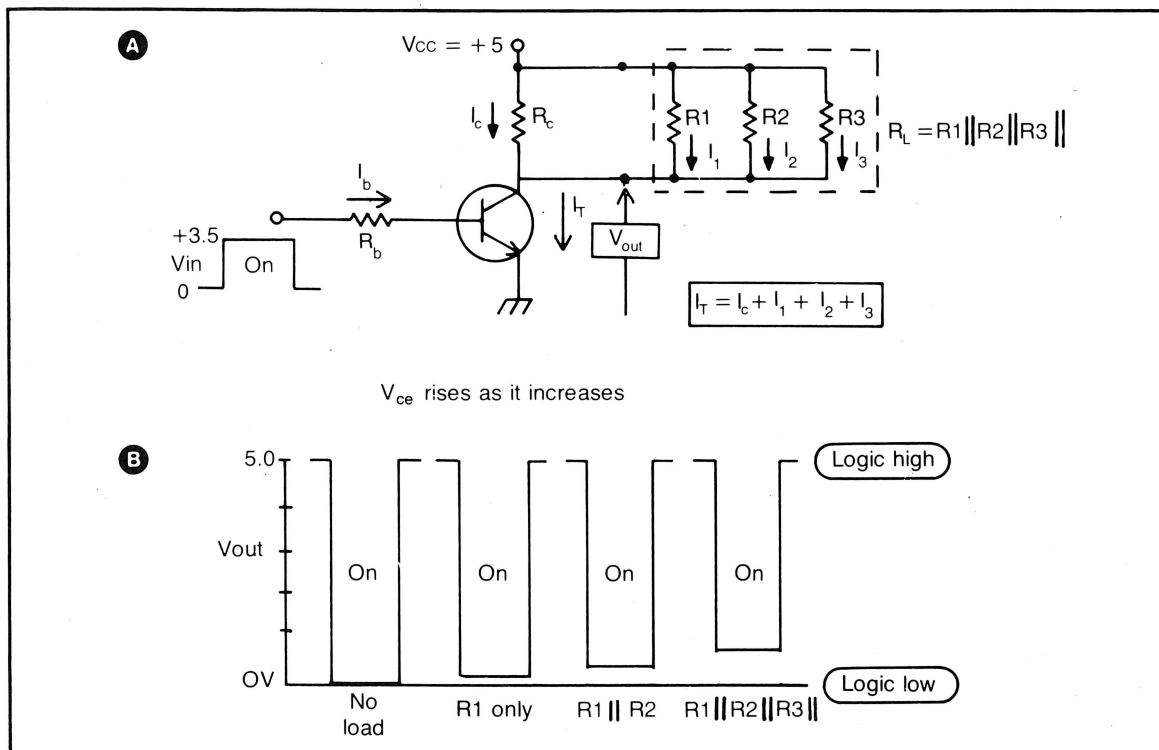
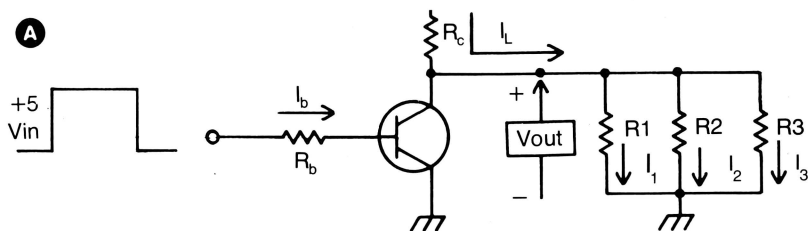


Fig. 6-26. Excessive loading in a current sink configuration. A) Example circuit. I_T is the total collector current, and increases as parallel loads are added. B) With excessive collector current, the transistor comes out of saturation and $V_{ce, low}$ rises.



$$R_L = R1 \parallel R2 \parallel R3$$

$$V_{out} = \left(\frac{R_L}{R_L + R_c} \right) V_{cc}$$

V_{out} falls as R_L decreases

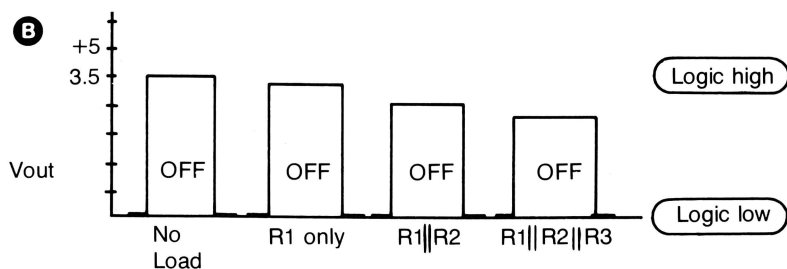


Fig. 6-27. A) Excessive loading in a current source configuration. A) Example circuit. B) $V_{ce,high}$ falls with excessive loading. This follows from simple voltage division.

Further, the output of a TTL IC is nothing more than a miniature transistor fabricated on silicon. Because of their design, TTL devices are intended to be connected in a current sink configuration. So, another practical implication of this discussion concerns not the individual transistor as a separate component, but rather as part of a digital integrated circuit: If you attempt to drive too many IC devices from a given TTL output (excessive loading), you are liable to get out logic lows which are too high. The output high's are not affected by excessive loading. But such loading may result in output lows that are in the indeterminate range for TTL input lows, that is above 0.8 volts. As you will

see in the next chapter, the amount of loading that a given device can withstand is expressed in terms of its current delivery capability.

Of course, there is a related, though complementary, problem of excessive loading with current source circuits. In the circuit of Fig. 6-27A, the loads are shunted between the output and ground across the transistor. V_{out} is the same as V_{ce} , as before. Also, we assume that for some specified value of R_c , the transistor is fully saturated when V_{in} is at +5 volts.

When the transistor is conducting in saturation, the current path is through R_c and through the collector-emitter circuit, completely bypassing the

load resistance. In other words, whatever the value of R_L (provided it is a few tens of ohms or greater) most of the current will flow through the near zero resistance collector-emitter path.

Now let us say that the transistor is off ($V_{in}=0$). With the transistor presenting an ideally infinite resistance, all the current is shunted through R_L . V_{out} is now at some logic high value, one that is determined by simple voltage division. We assume that the circuit was originally designed for only R_1 in place, and that the resultant output high would be 3.5 volts. However, as we add more resistive loads in parallel with the first, the equivalent load resistance will decrease. Consequently, the voltage drop across R_L , which is V_{out} , will decrease. We can summarize this by saying that overloading a current source configuration results in falling logic high levels.

This decrease in the logic high level with excessive load currents is illustrated in Fig. 6-27B. Again, note that the logic low level is not affected.

The implications of the overloaded source are the same as for the overloaded sink. When using discrete transistors, compatibility with TTL levels should be assured by avoiding excessive load currents lest the output high fall below the 2.4 volt minimum for TTL output logic levels. As regards digital ICs, one must be sure not to connect too many devices in source configuration, for the same reason. It should be emphasized that TTL is not normally connected in source configuration to provide driving current, but rather in sink configuration. This is because the TTL family has been designed to provide more current driving capability in sink than in source configuration.

With other families of digital devices, such as CMOS, these same generalizations are true. Drawing too much source current decreases the output high logic level, and drawing too much sink current increases the output low voltage level.

Transistor Ratings

The ratings we are concerned with in our applications are given below. They are much simpler than if we were involved with analog applications, where considerations of thermal stability, linearity,

accurate biasing, bandwidth and frequency response, distortion and other factors would be important.

For a typical general purpose transistor, the 2N2222, the following ratings apply:

Polarity—npn

Power dissipation—1800 mW or 1.8 watts.

Taken as the collector current times the voltage across the transistor: $I_c \times V_{ce}$.

$I_{c_{max}}$ —1800 mA. Not to be confused with the saturation current. $I_{c_{max}}$ is the maximum current allowable, whether V_{ce} is zero or not. Saturation current can, of course, be much less than $I_{c_{max}}$, depending on the resistances in the CE circuit.

Current gain or beta—200. This is the slope of the current transfer curve, given approximately by ratio of collector to emitter current, I_c/I_b .

$V_{ce_{max}}$ —30 volts. This is the maximum voltage which may be placed across the collector and emitter leads. In effect, it is the maximum supply voltage, because when the transistor is off, this is the voltage that appears across these leads. Beyond this voltage, the transistor would break down.

Frequency-gain product—300 Megahertz. Gain falls off at very high frequencies. A gain-frequency product (gain-bandwidth product or f_t) of 300 means that the transistor can operate at switching speeds up to 300 MHz. However, at this frequency, the gain will be unity (no amplification of current). At 30 and 3 MHz, the beta will be 10 and 100 respectively. At 1.5 MHz the gain will be 200, and below this frequency, the beta will remain at this nominal value. In other words, you have a compromise situation with any transistor—if you operate at higher frequencies, the current gain is less, and vice-versa. At the several MHz frequencies employed in microcomputers, such fall-offs in the gain of general purpose transistors are usually minimal.

Summary

One simple way to remember the important switch properties of the transistor is to recall the current and voltage relationships between emitter, collector, and base, and the fact that the transistor is a current amplifying device. To this, add the conditions for the on and off states.

Current Relationships: From Kirchhoff's Current Law and from the current amplifying ability of the transistor we have:

1. $I_e = I_b + I_c$
2. Beta or current gain $= I_c / I_b$ and, because I_b is relatively small,
3. $I_e = I_c$ or approximately so.

Voltage Relationships: From Kirchhoff's Voltage Law, the interlead voltage relationship is:

1. $V_{ce} = V_{cb} + V_{be}$ and since in forward bias $V_{be} = 0.7 \text{ V}$,
2. $V_{ce} = V_{cb} + 0.7 \text{ V}$.

Off or Nonconducting State: The transistor, or more accurately the C_E circuit is an infinite resistance, and all of the supply voltage appears across it. Nil current flows through the base-emitter junction; it is unbiased.

1. $V_{ce} = V_{cc}$.
2. R_{eq} of transistor is very high.

On Saturated, or Fully Conducting State: The transistor BE junction is fully biased and maximum current flows in the CE circuit, limited only by the resistances present in the CE branch. The transistor (collector-emitter branch) presents near zero resistance.

1. $V_{ce, sat} = 0$
2. The saturation current which flows is generally given by, $I_{c, sat} = V_{cc} / [\text{total resistance in CE circuit}]$
3. The base current required for saturation is calculated by, $I_b = I_{c, sat} / [\text{beta}]$.
4. The value of the base resistor is given by $R_b = (V_{in} - V_{be} / I_b)$.

Naturally, one may decide that it is the zero voltage, or alternatively, that it is a specific value of current, that is important, as in the shunt and series examples given earlier. Also, the presence of diodes in the CE-circuit requires that the forward bias voltage drop be taken into consideration in the equation for $I_{c, sat}$.

A powerful and very economical model of the transistor (for our limited purposes) is given by the

variable resistor model. Just think of the transistor as a sort of potentiometer with current gain.

The transistor as switch can act as an inverter (shunt load or series collector load) or as a noninverting switch (series load in the emitter leg, or emitter-follower).

Overloading causes depression of the logic high in the current source or shunt configuration, and elevation of the logic low in the current sink or series configuration.

Finally, a certain delay in rise and fall times is due to the pn junction internal capacitance. When several transistors are present in a circuit, the total effect of such capacitive lags is cumulative. As a result, an effective *propagation delay* from high to low or from low to high transitions at the output is evident. Thus the speed of signal transmission though a multitransistor device, such as a digital IC, is dependent upon this small, internal (junctional) capacitance.

EXPERIMENT 9A, DIODE AND TRANSISTOR TESTING

Purpose

To test the integrity of diodes and transistors and to distinguish between npn and pnp transistors.

Materials

General purpose diode (RS 276-1104) or equivalent

General purpose npn—2N2222 or equivalent (RS 276-1617).
pnp—2N3906 or equivalent (RS 267-1604).

Procedure

1. We will use a voltmeter to test the forward and reverse bias resistance of a diode, as illustrated in Fig. 6-28A and B. The resistor testing circuit inside the VOM or DMM includes a battery; the amount of current through the tested component determines the resistance of that component and is reflected by the meter reading. The test lead of the voltmeter is usually connected to the positive terminal of this battery, the ground or common lead to the negative terminal. (Rarely, this connection is reversed, but

there is an easy way to find out: examine the schematic diagram that comes with the instrument.)

2. Connect the (assumed) positive test lead to the anode and the negative ground lead to the cathode. Remember, the cathode is the banded side of the component. Most of the voltage ranges in the voltmeter employ a battery source of about 1.5 volts or so. This is enough to forward-bias the diode, and give an expected low reading for resistance, as in Fig. 6-28A. This value varies widely, but in any event, it should fall within the range of 100 to at most a few thousand ohms.

If you do not get the low reading try different voltage ranges, or consult the instrument manual regarding diode testing. Some DMMs have special high and low ranges specifically intended for diode and transistor testing. If you still get a high resistance value, try reversing the leads, in the event that the battery polarity is reversed from the conventional arrangement. If you still get a very high resistance value, try replacing the battery.

3. Now reverse the leads, with the positive test lead connected to the cathode and the negative ground lead to the anode, as in Fig. 6-28B. You should have a high resistance value, probably in the 100 k ohm to several megohm range. This is expected for a reverse-biased diode.

Once you have your lead polarities sorted out (and perhaps a fresh battery), you can test any pn junction in a similar fashion. Occasionally, you will find an open circuit when the diode is forward-biased, or a short circuit or very low resistance when it is reverse-biased. In either case, the component is obviously no good. This examination of a diode's integrity is gross, but does provide a very useful go/no-go test of function.

4. In a similar manner, you can test what type of transistor you have, pnp or npn, by applying the diode testing procedure outlined above. A typical plastic case for a general purpose transistor, such as the 2N2222 or 2N3906, is shown in Fig. 6-28C. The emitter, base, and collector are identified on

the bottom view of the component. Generally, the base is found as the central lead. By connecting the voltmeter leads as shown to an npn (2N2222) transistor, a low resistance value will be obtained. This is indicated in the figure by the battery circuit in the voltmeter which forward-biases the base-emitter junction. The collector-base junction can be tested in a similar fashion.

Caution: Avoid the lowest resistance range

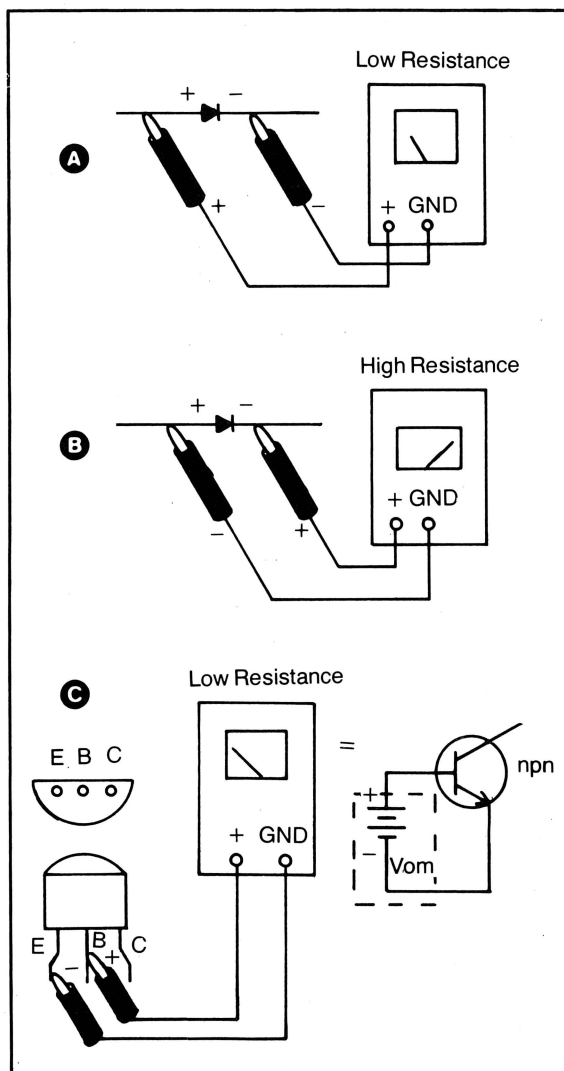


Fig. 6-28. A and B) Diode testing. C) Testing an npn base-emitter junction. See Experiment 9, step 4.

when forward biasing transistor junctions. This prevents excessive current which may (though usually does not) do damage to the junction.

5. Test a pnp transistor, such as the 2N3906, in the same manner.

EXPERIMENT 9B, TRANSISTOR CURRENT GAIN AND THE OVERLOADED SINK

Purpose

To demonstrate a quick and fairly accurate way of measuring a transistor's current gain and to demonstrate what happens with overloading in the sink configuration. To demonstrate the necessity for underestimating this gain in order to achieve saturation. To show what happens to V_{ce} when a sink configuration is overloaded.

Materials

- 1 - 2N2222 npn transistor (RS 276-1617)
- 1 - 150 K resistor
- 1 - 1 K resistor
- 1 - 0-5000 ohm potentiometer (RS 271-210)

Procedure

1. Wire up the circuit shown in Fig. 6-29A on the solderless breadboard, with the computer off. It is best if you bought a number of transistors of the same serial number, such as supplied by Radio Shack or other discount suppliers. This will allow you to see how much the current gain varies from the listed value. Attach two 4 or 5-inch leads to the potentiometer (or pot) and attach it as indicated in the figure. Now connect the power and ground from the Apple Game Socket (pins 1 and 8 respectively), and simply turn on the computer.

2. Attach the leads of a voltmeter across the collector and emitter, so that you can measure V_{ce} . Then adjust the pot to maximum so that the collector current I_c is at a minimum. (A clockwise rotation should increase potentiometer resistance if it is connected as shown; verify that this is so beforehand, by checking pot resistance out of circuit.) The value for V_{ce} should be very near zero (perhaps 0.08 to 0.14 volts).

3. Now gradually decrease pot resistance R_p by turning the shaft counterclockwise. By decreasing pot resistance, you are decreasing the resistance in the collector circuit. Consequently, I_c increases. Provided that the current fixed in the BE circuit is enough to saturate the transistor, V_{ce} will remain close to zero, say 0.12 volts. At some point, as you decrease R_p , and thereby increase I_c , the voltage V_{ce} will increase slightly. As you continue to do so, V_{ce} will increase more rapidly: Essentially, by drawing excessive current in the collector circuit, we have caused the transistor to fall out of saturation.

4. Vary the pot back and forth between saturation (minimum CE voltage) and slight desaturation—where V_{ce} is a few hundredths or maybe one-tenth of a volt above the minimum. In other words, find the approximate knee in the I_c versus V_{ce} curve, as indicated in Fig. 6-29B.

5. This knee in the curve represents the point at which the transistor just about saturates, given some fixed value of I_b . The voltage you measure is approximately equal to $V_{ce,sat}$ or perhaps slightly above it. Now, to measure the beta of this transistor first calculate $I_{c,sat}$:

$$I_{c,sat} = (V_{cc} - V_{ce,sat}) / (R_c + R_p)$$

where $V_{cc} = 5 \text{ V}$.

$V_{ce,sat}$ (approx.) might be around 0.2 V.

$R_c = 1 \text{ K}$.

R_p is determined by direct measurement, out of circuit.

Note: When you've adjusted the pot so that the transistor is just barely out of saturation—at the knee of the curve—simply remove it and use your DMM to measure its resistance to obtain R_p .

6. Calculate the actual or true beta of this particular transistor by using the equation in the figure. Note that the value of I_b can be verified by actually measuring the voltage across R_b (about 4.3 V) and dividing by its actual resistance value (which should be within 5% of 150 K).

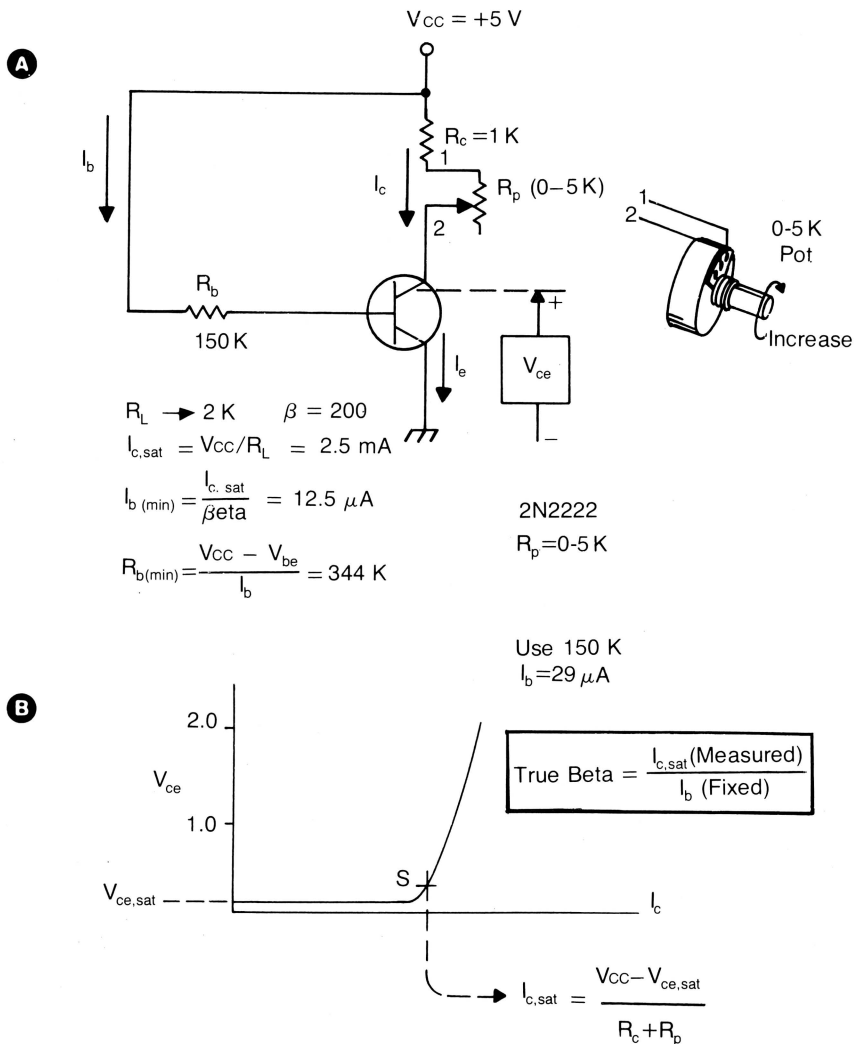


Fig. 6-29. Testing the beta of an npn transistor. A) Circuit set-up. B) Graphic relation of V_{ce} and I_c . Refer to the text for this experiment.

$$\text{True beta} = \frac{I_{c,\text{sat}}}{I_b} \text{ (measured in step 5)}$$

$$I_b \text{ (fixed by calculations in Fig. 6-29A)}$$

7. You should repeat this process for several transistors of the same part number. Vary the pot to cause V_{ce} to rise very slightly above saturation, then measure the value of the pot, calculate the approximate $I_{c,\text{sat}}$, and finally calculate the true beta.

Discussion

If you think carefully about this little demonstration, you will discover that it illustrates a number of important transistor properties.

First, you constructed a common emitter circuit with a fixed base current. This fixed I_b was determined from figuring 1) what current would be flowing at saturation for a 2 K load, 2) calculating I_b

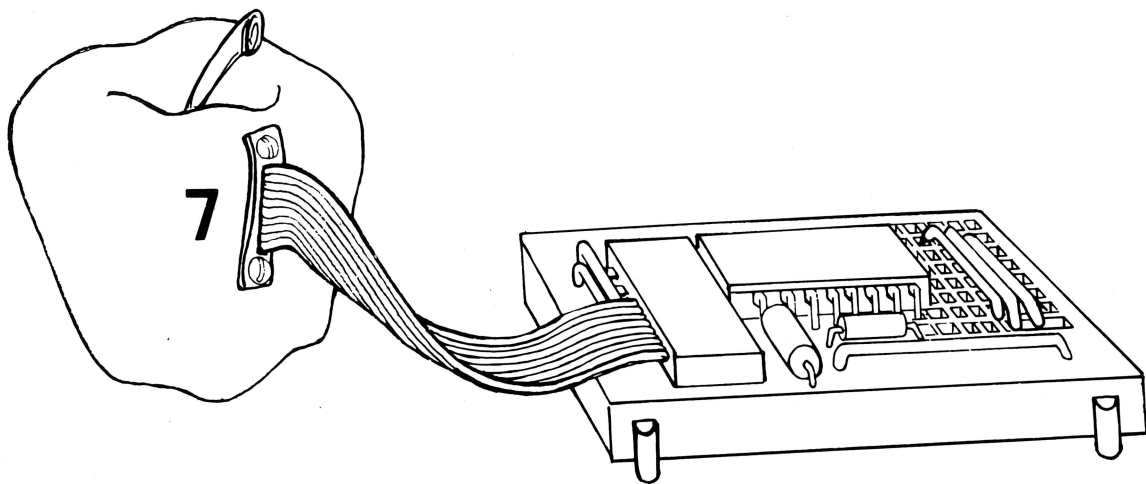
using the nominal value of beta for 2N2222, and then 3) calculating the base resistor R_b . To be generous (pessimistic actually), we then cut R_b from its ideal value of 344 K down to 150 K. This provided more than twice the minimum base current—29 μA instead of 12.5 μA . The bottom line to this over-biasing is that we are anticipating that the beta may be less than the manufacturer's specification, and are providing enough current to drive a transistor with a beta as low as 80 into saturation.

By varying the pot we are simulating an increasing load or number of loads, as might occur when either driving a more current-hungry device, or a greater number of devices. The increasing load is in sink (series) configuration. The transistor comes out of saturation due to insufficient base current (which is fixed, remember) as R_p is decreased. V_{ce} rises at this point, and in fact continues to rise linearly as I_c increases. By taking V_{ce} slightly above the minimum (saturation) voltage, we have a good approximation of $V_{ce,sat}$ and $I_{c,sat}$. $I_{c,sat}$ is calculated by taking the voltage drop across the collector-emitter resistance, and dividing the amount of that resistance, as given in the equation. (We include this measured value of V_{ce} , to gain a bit more accuracy.)

Beta is calculated by the ratio $I_{c,sat}$ (measured) / I_b , and is a good approximation to the transistor's true current gain. If you measure several transistors, you will probably find that the beta will vary from about the specified value, down to perhaps $\frac{1}{3}$ of this value. In fact, some of the transistors may not saturate fully with the 150 K base resistor; you may measure a V_{ce} of .2 or .3 volts even with minimum I_c (R_p adjusted to a full 5 K). Therefore, you may want to substitute a 100 K resistor for R_b , and use 43 μA as the basis for calculating beta. One series of measurements of about 15 or so 2N2222 transistors revealed betas ranging from 64 to 202 with the average about 85 to 100. This is less than $\frac{1}{2}$ the listed current gain!

This brings us to the second point of the experiment. Always overdrive the transistor with extra current. Up to two or even three times the minimum I_b calculated is suggested. This is particularly important with surplus components.

Third, you have a very clear demonstration of the rising logic low with the overloaded sink. This same problem occurs with any TTL IC device. The practical necessity of avoiding invalid logic levels by excessive overloading is obvious, and from this project you should have gained a physical understanding of how such invalid lows can come about.



TTL Internals

In this final chapter in the Essential Electronics unit, we'll examine the digital IC not as a Boolean device performing logical functions, but rather as an electronic circuit with certain electrical characteristics. You've already been introduced to the idea of TTL voltage levels for logic high and low in Chapter 2. But here, the whole set of device parameters—voltages, currents, speed, noise, power requirements, etc—will be covered in some detail.

The simple logic gates that perform Boolean operations were earlier examined as functional black boxes. Logical variables and values were assigned to the voltage levels at the input and output terminals—logic high and low, true and false, etc. You saw how Boolean (combinational logic) operations were executed without regard for how the device actually worked. This was convenient, because it allowed you to hook up these IC building blocks using the laws of Boolean algebra, just as if they were terms in a Boolean equation. The marvel of IC technology is that you can even construct sophisticated circuit functions while being rela-

tively ignorant of the electronics involved.

However, the real world does impose practical limits, and you must know something about the insides of digital ICs to use them correctly in working circuits. Now that you've had a sort of crash course in basic electronics, you can approach the digital IC as an electrical black box: a box with inputs and outputs whose properties are not Boolean values but voltages and currents. You will be able to draw on the terminology and concepts of the last two chapters to understand the electronics of a typical TTL IC. The goal of this chapter is to use this knowledge to gain a firm grasp of IC electrical parameters, and so to be able to use the wealth of technical data available on these devices.

Get out your TTL Data Manual (from any manufacturer) and start thumbing through the technical specifications as you read this chapter. The format of presentation varies slightly from one manufacturer's data book to another, but once you're comfortable with one, you'll be able to read and use any of the others with facility.

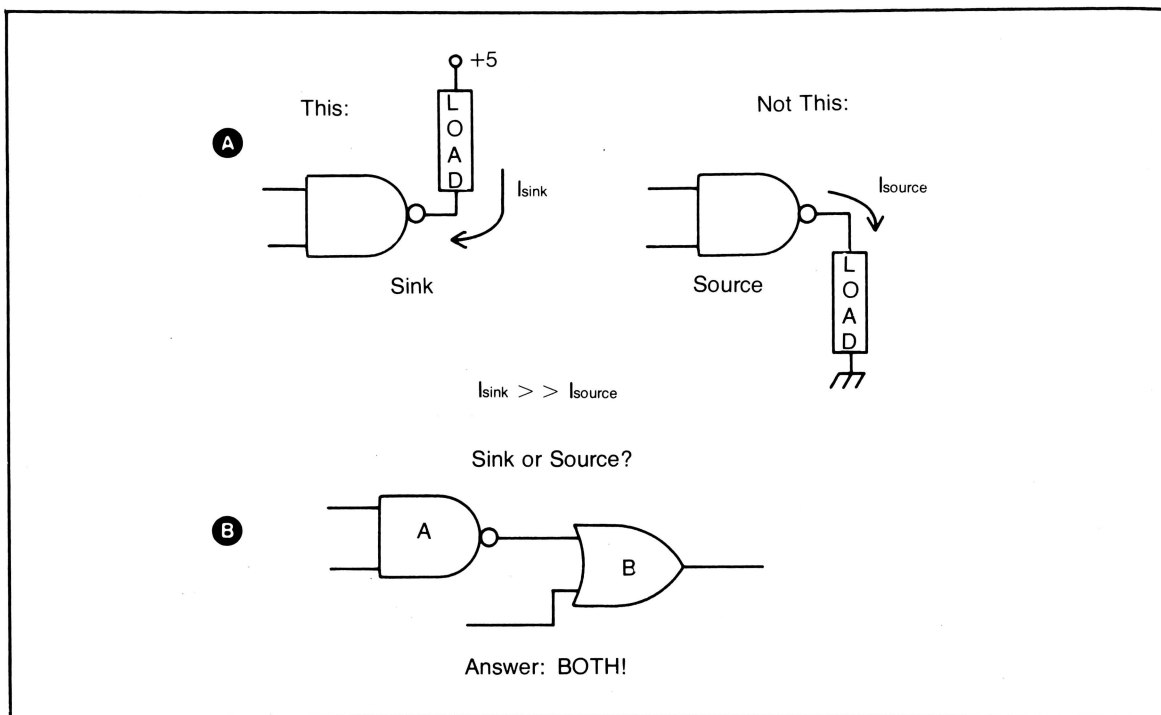


Fig. 7-1. A) TTL's sink current is greater than its source current rating. B) TTL to TTL connection—device A acts alternatively as sink and source depending on its output.

THE ELECTRICAL BLACK BOX

At this point, I want to present some electrical information on the internals of IC packages. Two important concepts are source and sink. It is also important to understand the input and output parameters.

The Source and Sink Approach

One way to grab hold of a complex topic is to ask a key question. In the case of IC electrical characteristics, the question we'll start with is: "What does the idea of current sourcing and current sinking have to do with digital ICs?"

TTL circuits are designed to provide about twenty times as much sink current as source current. Occasionally, you will want to drive a non-TTL device or circuit which consumes a fair amount of current relative to a TTL input. Therefore, when driving relatively high current-consuming loads (lamps, speakers, transmissions lines), those loads

are best connected in a sink rather than source configuration. This is suggested in Fig. 7-1A.

Notice that the direction of current flow is into the terminal of this NAND IC when it is sinking current, and OUT of the terminal when it is sourcing current. This is the same as with the transistor circuits of the last chapter.

Now suppose you connect one LSTTL device to another, as in Fig. 7-1B. Is device A serving as a current sink or as a current source for device B? The answer is both!

This is restated more explicitly in Fig. 7-2. When the output of device A is low as in 7-2A (both inputs tied to +5 volts), it is sinking current; current flows out of the input of the OR device and into the output terminal of the NAND gate. When the output of device A is high, as in 7-2A, it is sourcing current; the direction of current flow is out of the output terminal of the NAND and into the input terminal of the OR gate.

If you could describe how these sink and

source currents flow in a typical TTL circuit with sufficient depth and detail, you would by implication have a good grasp of device electrical parameters, and of the cardinal DOs and DON'Ts of using TTL devices as electrical components. In this chapter we will be using this unitary concept of sink and source to describe TTL operation and the major TTL characteristics. First, let's introduce some of the basic TTL parameters by reference to a black box model.

The Black Box Model of TTL

In the prior example in Fig. 7-2, we mentioned the currents into and out of the inputs and outputs of a TTL device. Naturally, there are also corresponding voltages at the output and input terminals for the logic high and low states. Both currents and voltages in these states must meet the standards of the TTL family.

Figure 7-3 summarizes the key parameters of TTL with which we are concerned. A digital device is presented as a black box, with certain terminal characteristics—primarily the voltage and current

at the input and output and terminals. Certain ranges of input voltage and current must be maintained in both the logic high and low states. The same goes for the output voltages and currents. From these considerations alone, we already have eight parameters, as indicated in the figure.

In addition, there are related parameters, as well as design factors, to consider. The major factors are listed in the figure. Some of these parameters have to do with the power requirements of an IC, and with its ability to provide driving current for devices in the same family. Dynamic factors involve noise generation (switching transients) and the overall speed of the device. There are also circuit design considerations which either improve on one or more of these factors of which add some totally new feature.

As I detail some of the nuts and bolts of TTL circuitry, you might want to refer to this figure in order to maintain the overall perspective. Remember, the point is not to be able to draw complex internal circuitry from memory but to gain a physical understanding of what the various parameters mean.

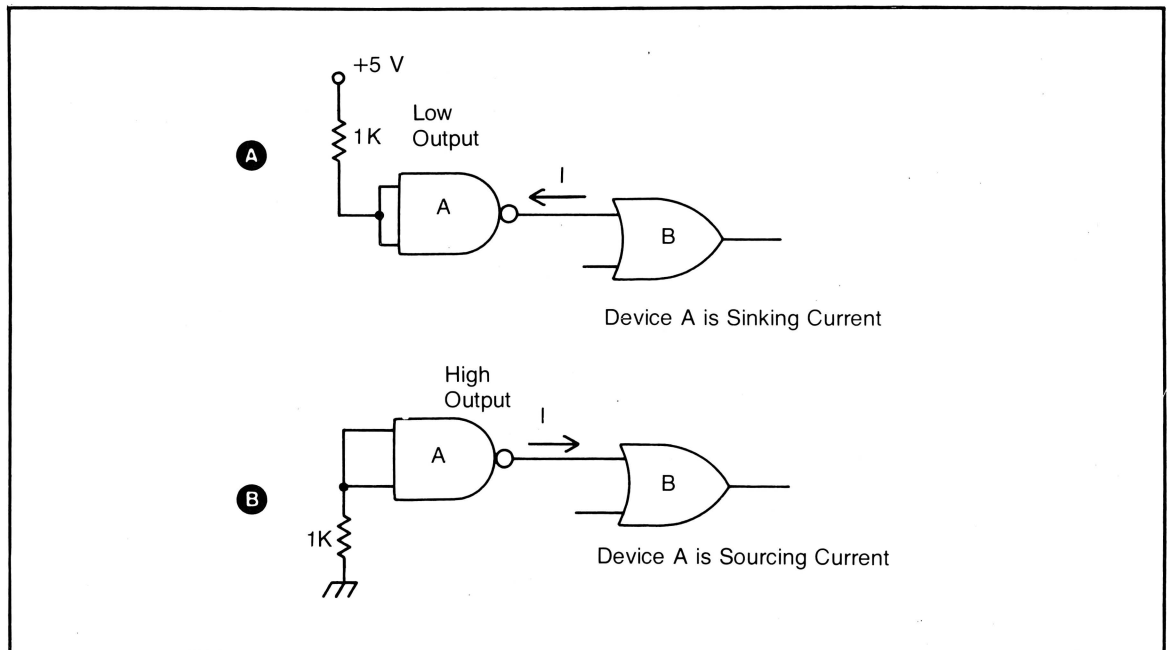


Fig. 7-2. Direction of current flow in output low/sink and output high/source states.

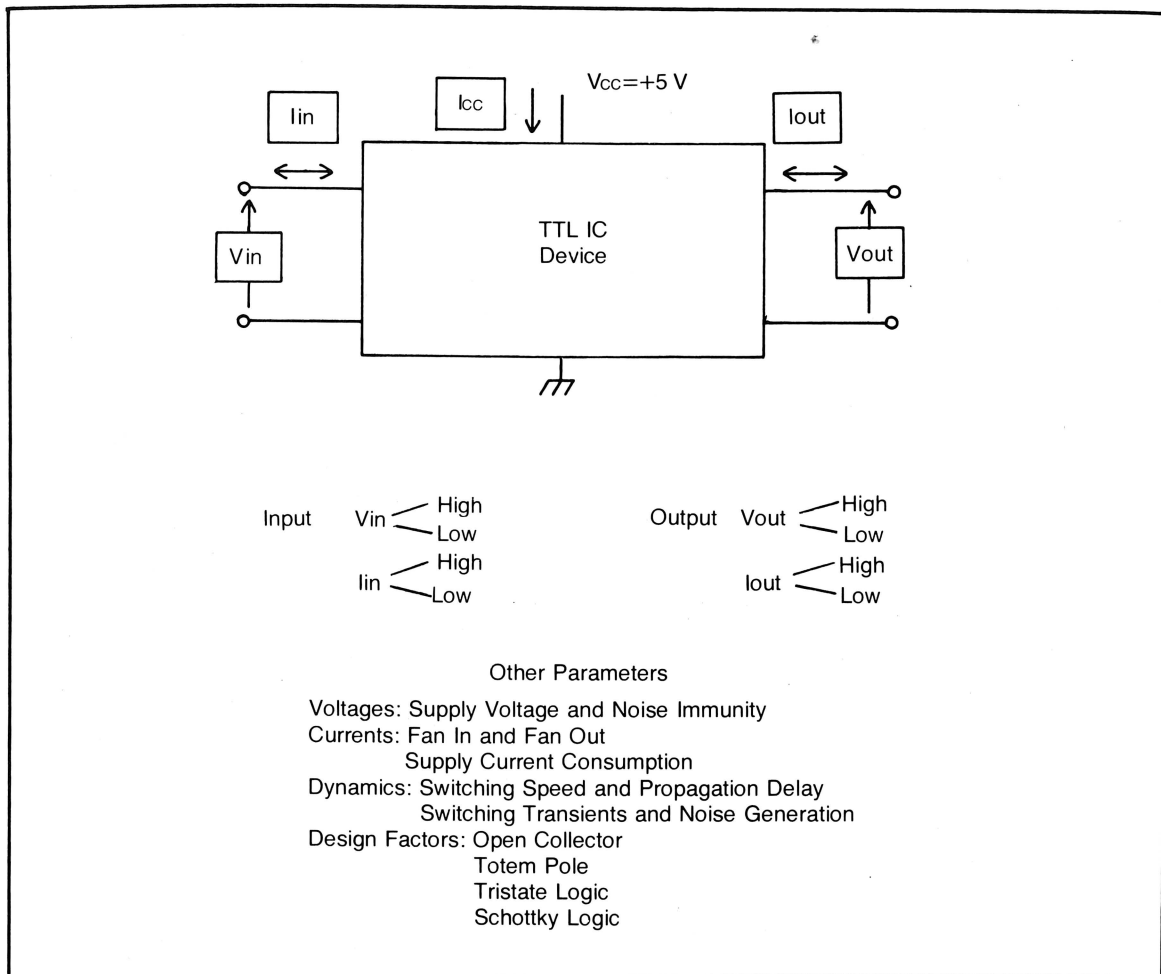


Fig. 7-3. Overview of major TTL parameters.

INSIDE THE BLACK BOX

Two major design approaches are used in ICs: open collector outputs and totem pole outputs. The circuitry for each approach is explained.

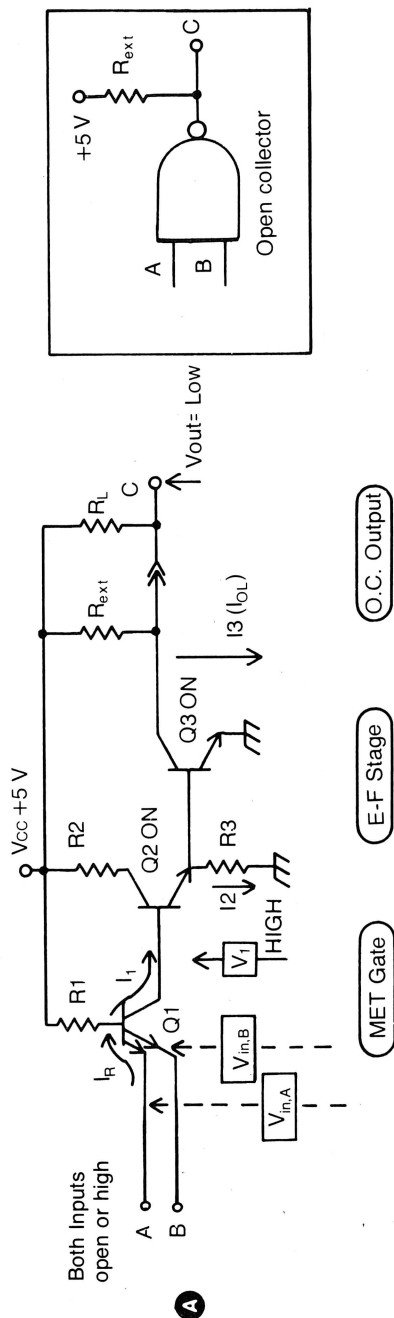
Open Collector Operation

The simplest TTL circuit we'll talk about is the three transistor NAND gate of Fig. 7-4. The discussion centers about eight parameters: the input and output voltages and currents for both the high and low output states. The transistors Q1-Q3 turn either on or off during circuit operation. You may assume that the values of the resistors have been

chosen so that when these transistors are on they are saturated (V_{ce} near zero).

This typical TTL circuit is organized in three parts. There is a multiple emitter transistor (MET), Q1, consisting of two or more emitters on the same transistor, each serving as a separate input. The EMITTER-FOLLOWER stage, Q2 drives output transistor Q3. Finally, the output is taken off the collector of Q3, a so-called *open collector* output. Let's see how this circuit operates for the two possible output states, high and low.

Output Low. We will first examine what is going on in the circuit when V_{out} is near zero, logic low.



A = Low
B = High

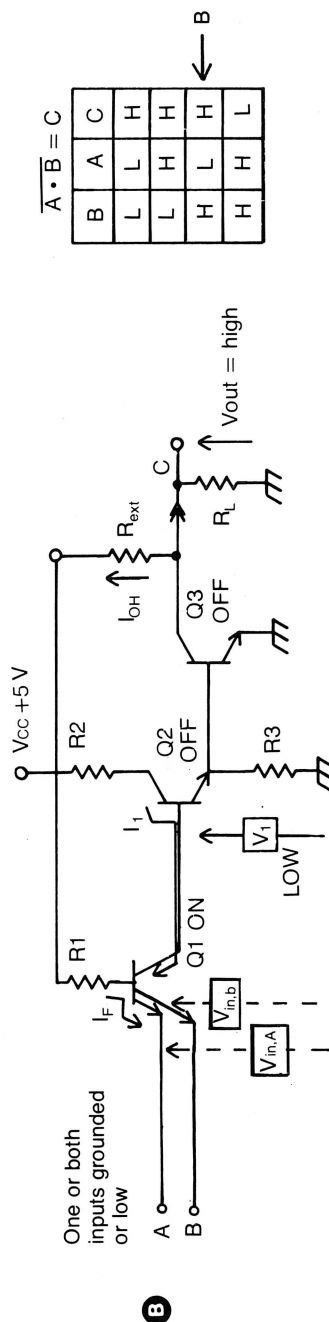


Fig. 7-4. Operation of a three-transistor, open collector NAND gate in A) output low and B) output high states. Arrow in truth table indicates the conditions in (B). An external resistor is needed in open collector devices.

It is important to note that a TTL device with a logic low output acts as a current sink with respect to the load (R_{ext} and R_L in Fig. 7-4A).

To begin, both inputs are tied to a positive V_{in} of, say, 3 to 5 volts. Because of the way Q1 is biased by base resistor R1, Q1 does not conduct. (One or the other emitter in this MET must in fact be near zero for the base-emitter junction to be forward-biased). This being so, the collector of Q1 is high, slightly less than V_{CC} . This voltage, V_1 in the figure, forward biases the base-emitter of Q2. With Q2 on (saturated and conducting), a current I_2 flows. This current causes a voltage drop across the emitter resistor R3, a voltage which serves to forward-bias Q3. Because Q3 is on (saturated and conducting), its collector-emitter voltage is near zero. That is, V_{out} is at logic low.

What about the input and output currents in this output low state? Because both inputs are at logic high, the BE junction of Q1 is nonbiased. Only a small minority current flows, one which is in the reverse direction to current flow in a forward-biased BE junction. This reverse current, I_R , is on the order of 20 to 40 microamps, and is sufficient to keep Q1 turned off. Phrased another way, the high voltage at the inputs serves to inject a wrong-way current into Q1's emitter, thereby assuring non-conduction. The current that flows from Q1's collector is I_1 , the magnitude of which is determined mainly by the values of the resistors in the conducting path from V_{CC} to ground, namely, R1 and R3. This current is small, but is in the proper direction to forward bias the BE junction of Q2, turning it on. Current I_2 flows through Q2. (I_2 is really Q2's emitter current, and is the sum of the Q2's base current I_1 plus the Q2's collector current). Current I_2 causes the voltage drop across R3, and turns Q3 on, as mentioned. The current through saturated Q3, I_3 , is variable, depending on the resistance in Q3's collector circuit. I_3 is the saturation current of Q3, and because of its direction of flow, it is a sink current.

Now, what about the limitations on these input and output voltages and currents? If V_{in} is too low, insufficient reverse current (I_R) will be injected into Q1's emitter. Q1 may or may not conduct. There-

fore, insufficient input current or voltage constitutes an *indeterminate input*, and is undesirable. At the output end, we must talk about a proper V_{out} . It must be within a few tenths of a volt of zero to be a valid logic low. If the load on Q3's collector is excessive (if collector load resistance R_L is too low and hence draws too much current) then Q3 will come out of saturation. V_{out} will consequently rise, and V_{out} will no longer be a valid low.

The standard limits on input high voltage and current and output low voltage and current for TTL are as follows:

V_{IH} (min input high voltage) = 2.0 V. This corresponds to V_{in} , A and V_{in} , B. At least 2.0 volts is required for a standard (STD) or low-power Schottky (LS) input to sense a logic high.

I_{IH} (max input high current) = 20 or 40 μA . This corresponds to I_R . This is the most current that would be required for LSTTL (20 μA) and STD-TTL (40 μA) inputs to sense a logic high.

V_{OL} (max output low voltage) = 0.4 V. That is, V_{out} . All TTL devices are guaranteed to provide a maximum of 0.4 volts as an output logic low under normal loading conditions.

I_{OL} (max output low current) = 8 or 16 mA. LSTTL and STD-TTL devices are designed to provide a nominal value of output low or sink current of 8 and 16 mA respectively. If you attempt to draw more than this, V_{OL} will rise due to desaturation (overloaded sink).

Output High. The state of affairs in a TTL NAND gate with an output high is easily understood if you followed the above. This condition is shown in Fig. 7-4B.

Important to note is that in the output high state, a TTL device acts as a current source with respect to the load (R_L in Fig. 7-4B).

If one or both inputs is at logic low, then the BE junction of Q1 is forward-biased. Q1 is saturated/on, and its collector voltage (V_1) is low. Q2 is therefore off/nonconducting, and negligible current flows through R3. With near zero voltage drop across R3, BE junction of Q3 is not biased, and Q3 is off/nonconducting as well. V_{out} , the collector-emitter voltage of Q3, is therefore high.

Current flow in this output high state is as follows. With one or both Q1 emitters grounded, the forward-biased BE junction conducts a forward current I_F , the direction is opposite that of I_R in Fig. 7-4A. This is a significant current, on the order of a milliamp or so. Q1's collector current, I_C , is also flowing in the opposite direction from that in Fig. 7-4A. One can think of it as preventing Q2 from conducting because it is flowing in the wrong direction relative to Q2's BE junction. Since Q2 is OFF, so is Q3. The only current that could flow in the output circuit of Q3 is from a load resistor connected between the output and ground, R_L in the figure. This would be the $I[\text{output, high}]$ or I_{OH} source current.

The specifications for the currents and voltages described are as follows:

V_{IL} (max input low voltage) = 0.8 V. This is the maximum voltage allowed for TTL inputs to sense a logic low.

I_{IL} (max input low current) = 0.4 mA or 1.6 mA. This is the maximum current that would be necessary for an LS or STD TTL input to sense a logic low, respectively.

V_{OH} (min output high voltage) = 2.4 V. TTL outputs are guaranteed to provide at least this voltage for an output high.

I_{OH} (max output high current) = 400 μ A. LSTTL and STD-TTL are guaranteed to provide a nominal value of 400 μ A (0.4 milliamp) of source current in the high output state. If you attempt to draw more than this with too low a value of load resistance, then V_{out} will fall below the valid logic high level.

Another feature of this TTL gate is that when the output is low, Q1 is off and both Q2 and Q3 are on. When the output is high, Q1 is on and both Q2 and Q3 are off. This is important, because it means that this open collector NAND consumes more current when the output is low than when the output is high. The significance of this is probably obvious: There are more charge carriers flowing through BE junctions in the low output state; therefore, it takes longer for the device to switch from low to high than

from high to low. This is also generally true of other TTL configurations and subfamilies.

The truth table for this NAND device is shown in the inset of Fig. 7-4. The arrow indicates the condition described in 7-4B.

Also shown in the figure is the logic schematic symbol for this open-collector NAND. The open collector TTL, an early subfamily of TTL devices, requires an external resistor, R_{ext} . When connected between the output C and voltage source V_{CC} , R_{ext} serves as a pull-up resistor in the output high state. The value of R_{ext} must be low enough to assure that the output high voltage is at least 2.4 volts or greater. On the other hand, R_{ext} must be large enough to assure that the maximum sink current (I_{OL}) through Q3 is not exceeded in the output low state.

Totem Pole Operation

There are problems with open-collector ICs. First, the user must supply an additional component. The value of this external resistor, R_{ext} , must also be calculated to satisfy the loading conditions of the circuit in which the IC is used. R_{ext} usually works out to about 2 K or so for most applications.

But more serious than this inconvenience is that this added resistor slows down circuit operation. This resistor combines with the distributed capacitance in the circuit to form an RC time constant which significantly lengthens the transition or switching time. The transition time from low to high output is particularly affected because there are more charge carriers in the circuit, as just mentioned.

A solution to these problems of slow transition, extra calculation and an added component is the *totem-pole output*. Let's discuss the entire configuration.

Figure 7-5A and B illustrate the totem-pole TTL output, comprised of transistor Q4 and resistor R4. The input is Q1, the MET (multiple input transistor); Q2 is called the phase splitter transistor; and Q3 and Q4 constitute the output circuit. There are several embellishments to this TTL NAND device.

First, you'll note that the transistors are all

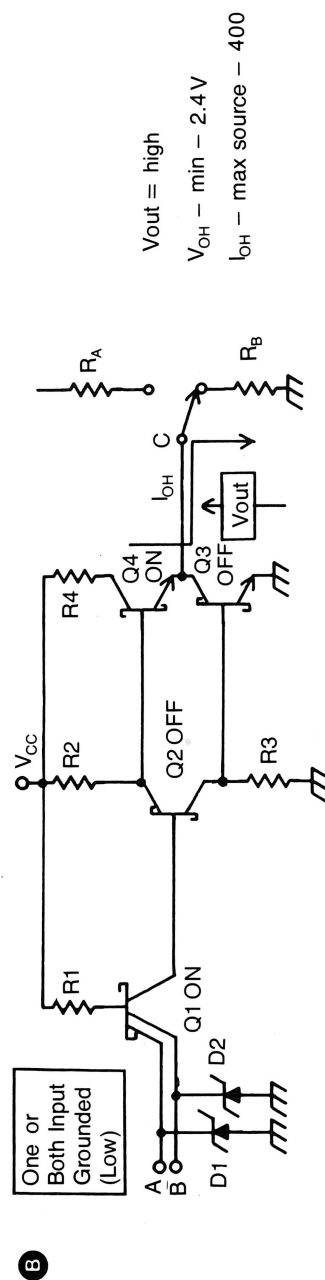
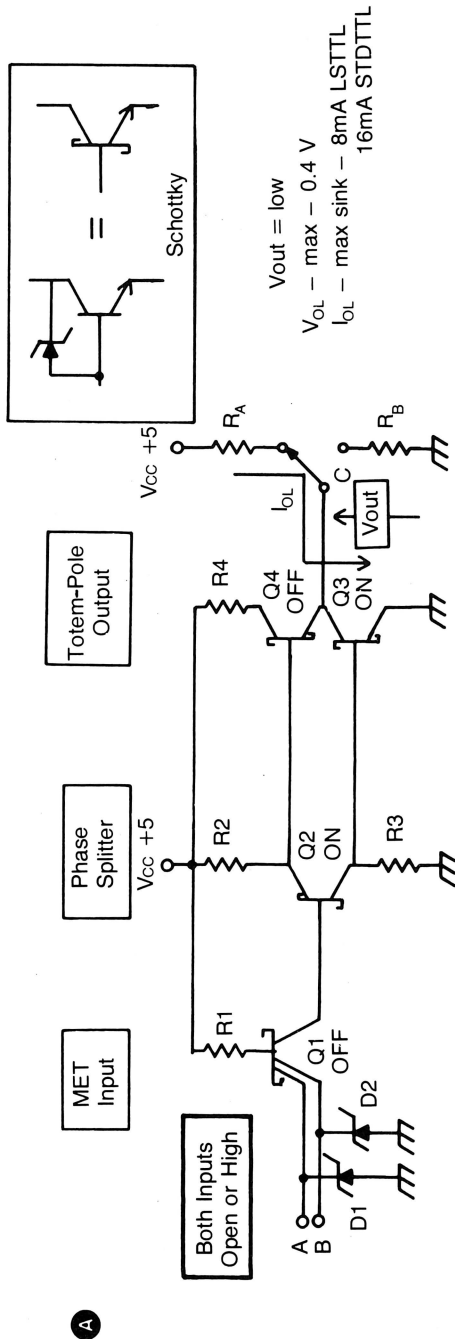


Fig. 7-5. Embellishments of the open collector configuration include a totem-pole output, input protection diodes, and Schottky clamped transistors. See text.

Schottky devices, as indicated by the insert in the figure. As discussed earlier, Schottky transistors when in the on/conducting state, are very close to but just short of total saturation. This improves switching time from on to off by avoiding excess charge carriers in the BE junction.

Second, the two input diodes, D1 and D2, serve to prevent damage to Q1 if the input voltage is less than zero. Excess negative voltage on either emitter of the MET might cause excess current flow through Q1's BE junction, thereby destroying the transistor. This is never the case in a well-designed circuit. However, negative voltage may be applied inadvertently, or through contact with nearby objects with static electrical charge. In either case, the diodes will conduct the charge to ground, harmlessly.

Third, there is the main feature of this circuit, the totem-pole output itself. The inclusion of transistor Q4 serves to improve switching time. Instead of a passive external resistor, as in the open-collector circuit, we have an active pull-up, a transistor, which can snap on and off very quickly. In particular, the low to high transition time is considerably improved with the totem-pole.

Output Low. In order to explain the operation of the totem-pole, the circuit is shown in its two output states. Refer to Fig. 7-5A. As in the previous open-collector circuit, Q1 is off because inputs A and B are high, and therefore Q2 is on/conducting. Transistor Q4 is held in the off/nonconducting state because the voltage at Q2's collector is near zero, due to its near-saturated state. Q3 is on/conducting due to current through R3. With Q3 on, its collector voltage, V_{out} , is near zero volts. The output at point C is then a logic low. The device acts as a current sink, with current flowing from the +5 volt supply, through load resistor R_A and then through Q3. This would be the desired configuration if the device were driving a relatively high current-consuming load (several mA or more).

Totem-pole devices are designed to operate with the same ratings as open-collector devices, because they are members of the same broad TTL family. Hence, provided that R_A is not too low, the maximum output low voltage (V_{OL}) will again be 0.4

volts. The maximum current allowed for this output low voltage rating (I_{OL}) is also 8 mA or 16 mA for LSTTL and STD-TTL, respectively. This means R_A can have a minimum value of about V_{CC}/I_{OL} . This works out to be 330 ohms for a STD-TTL device and 680 ohms for LSTTL, using commonly available values.

Note that this output low state can be achieved with TTL devices by simply leaving the inputs open, that is, unconnected. This leaves Q1 unbiased and off. However, in the final version of a circuit it is generally good policy to tie the inputs high through a 1 or 2 K resistor, as unconnected inputs tend to *float*—they act as tiny antenna and pick up circuit noise, resulting in spurious operation.

Output High. When one or both of the inputs is grounded, Q1 turns on, and the phase-splitter Q2 turns off. As a result, Q3 turns off and Q4 turns on. With Q3 presenting an extremely high resistance in its nonconducting state, most of the current will flow from V_{CC} , through R4 and Q4, and finally to ground through load resistance R_B . The device is acting as a current source with respect to this load, as it is connected between the output terminal C and ground. See Fig. 7-5B.

LSTTL follows the same specifications as its other TTL brethren. The device must supply at least 2.4 volts in this logic high state under normal loading conditions. Also, Q4 cannot source more than 400 μA . Q4 serves only to provide a low resistance current path for the output high state. It is not designed to carry large currents. (The reason for this is that it will normally be driving low current consuming loads, specifically, other TTL inputs. This will be explained shortly). Since R4 is low—about 50 to 130 ohms, depending on the subfamily— R_B is the determining factor in limiting the current flow to 400 μA . R_B would therefore have to be at least V_{CC}/I_{OH} , or roughly 12.5 K, which is a relatively high resistance. This analysis simply supports the categorical statement already made, namely, that because of their design TTL devices must be used in sink configuration if you want several mA or more of current.

Another point about TTL: The totem-pole

works in combination with the phase-splitter, Q2. To work properly, Q3 and Q4 should never be on simultaneously as this would result in current peaks during high to low transitions and vice versa. Ideally, Q3 and Q4 should be exchanging their respective on and off states instantaneously. In practice, this is not possible. However, TTL designers have been able (through appropriate fabrication techniques and by proper choice of values of R2 and R3) to give phase-splitter Q2 a sharp transition characteristic between on and off states. That is, Q2 will not gradually go from conducting to nonconducting state, but will do so very suddenly. This, in combination with the low resistance of Q4, allows for the snap-action between high and low states at the output.

In the data manuals and other technical literature, you will sometimes see the advantage of the totem-pole output described as follows: "The totem-pole transistor is an active pull-up for the output high state, providing a low impedance (resistance) drive as a current source. This significantly improves switching time, which might otherwise be degraded, as in the open-collector arrangement." Or words to that effect. Certainly, this succinct statement in engineering jargon might totally confuse the beginner, but should be quite clear to you now.

THE KEY PARAMETERS OF TTL

There are two major categories of parameters that are presented here. They are static and dynamic. A good understanding of both will pay good dividends in your design endeavors.

Static Parameters

Now that you know about the voltage and current ratings of TTL, we'll turn to two other static parameters, *fan-out* and *noise immunity*. You will also see how a TTL output serves as alternative source and sink relative to another TTL device.

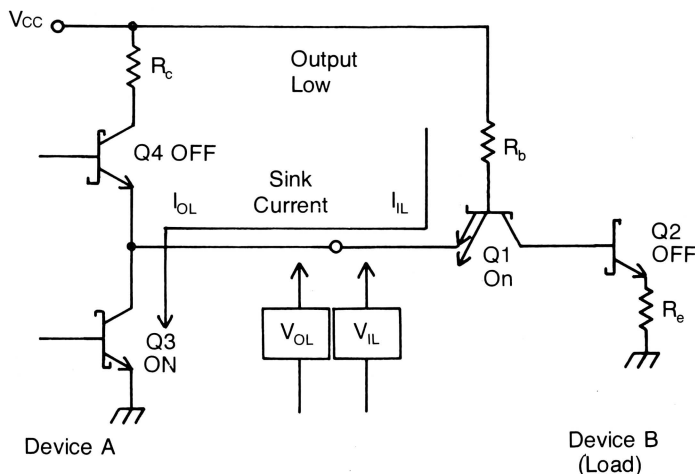
If you refer back to Figs. 7-1 and 7-2, you will remember the question we asked at the beginning of this chapter: How do you relate the sink and source configuration to TTL devices? In the preceding, external load resistors (R_A and R_B) were

used to show how current flows in the output high/source and output low/sink states. Implied in the explanatory figures (7-4 and 7-5) was an actual physical switching between two such external loads for each state. But just how does a TTL device act alternatively as either a sink or source in relation to another TTL device as indicated in Fig. 7-2? With a fair amount of material under your belt, you can probably answer this question yourself. Figures 7-6 and 7-7 give the answer explicitly. In the following, I will refer to LSTTL and STD-TTL parameters.

Output Low. Let's start with the output low state for a totem-pole Schottky device, the output of which is connected to the input of another similar device, as in Fig. 7-6. The situation is the same as that in Fig. 7-2B where device A is sinking current through its load, device B. Q3 in Fig. 7-6 is on and sinking current through its load. The load in this case is actually one of the BE junctions of the MET Q1 input, plus its base resistor R_b . Device A's output voltage will not exceed 0.4 volts, and device B will sense a valid low provided that the input voltage does not exceed 0.8 volts.

This means there is a 400 mV difference between the maximum allowable output low and input low (V_{OL} and V_{IL}). This voltage differential is referred to as the *noise immunity* of TTL devices, and is basically a built-in safety margin in the event of voltage transients. The noise immunity for logic low states is 400 mV. If a circuit transient of a few hundred millivolts occurs—a voltage spike that would briefly increase the output low of device A—then the input to device B will still be a valid low.

The current flow in this circuit is a sink current, as indicated in the figure. It flows in the direction to bias Q1 in the on/conducting state. LSTTL outputs are designed to provide (sink) up to 8 mA in the output low state. LSTTL inputs are designed to require no more than 400 μ A (0.4 mA) to sense a valid low input. This means that an LSTTL device can drive up to 20 other LSTTL devices. This number is obtained as a simple ratio of I_{OL} to I_{IL} . Because it is more convenient to talk of the number of devices that can be driven rather than the actual currents involved, this ratio is given a special name: *fan-out*.



$V_{OL,MAX}$	0.4 volts	} Noise Immunity 400 mV
$V_{IL,MIN}$	0.8 volts	

LSTTL {	$I_{OL,MAX}$	8 mA	Fan Out = $\frac{I_{OL}}{I_{IL}} = 20$ (for LSTTL)
	$I_{IL,MAX}$	400 μA	

$I_{CC,L}$	Supply Current when output is low.	2.4 mA per Pkg (LS00)
LSTTL		0.6 mA per gate $\frac{1}{4}$ LS00

Fig. 7-6. One TTL device in output low state driving another TTL input, with key parameters listed.

Fan out always implies that you are talking about a single subfamily. Another example: the fan-out of STD-TTL would be 16 mA/1.6 mA or 10. A standard TTL device will drive 10 other devices of the same subfamily. Obviously, the LSTTL subfamily has the added advantage of more relative drive power (fan-out) than STD-TTL, 20 versus 10.

Let me insert a note on the usage of the term fan-out; when mixing subfamilies the use of the term fan-out is a misnomer. For instance, you may

want to know how many standard TTL inputs could be driven by an LSTTL device. The answer is $I_{OL}(\text{LSTTL}) / I_{IL}(\text{STD-TTL}) = 8/1.6 = 5$ STD-TTL inputs. Conversely, a STD-TTL could supply enough sink current for $16/0.4 = 40$ LSTTL inputs. In such cases you must refer to the actual I_{OL} and I_{IL} involved for the respective subfamily, and take the ratio.

Also shown in Fig. 7-6 is the *supply current requirement* for LSTTL. When the output is low, a

typical LSTTL package (LS00 quad NAND used as an example) will require 2.4 mA. This works out to 0.6 mA per gate. Comparable ratings of STD-TTL are 12.0 and 3.0 mA of supply current per 7400 package and gate, respectively. Another advantage of LSTTL is, then, that of economy—these devices consume only about 1/5 the power of comparable STD-TTL devices!

Output High. Figure 7-7 illustrates one LSTTL circuit driving another for the output high state. Output high voltages for LSTTL devices are

guaranteed to be at least 2.7 volts or better, (For STD-TTL the figure is 2.4 volts). Inputs to all TTL devices, regardless of subfamily, must be at least 2.0 volts to be sensed as a valid logic high. Therefore, there is a 700 mV (0.7 volt) *noise immunity* for LSTTL. (For STD-TTL this safety margin is only 400 mV.).

Figure 7-8 summarizes the concept of noise immunity for both high and low states.

TTL outputs act as current sources in the high state. This current is shown in Fig. 7-7. It flows

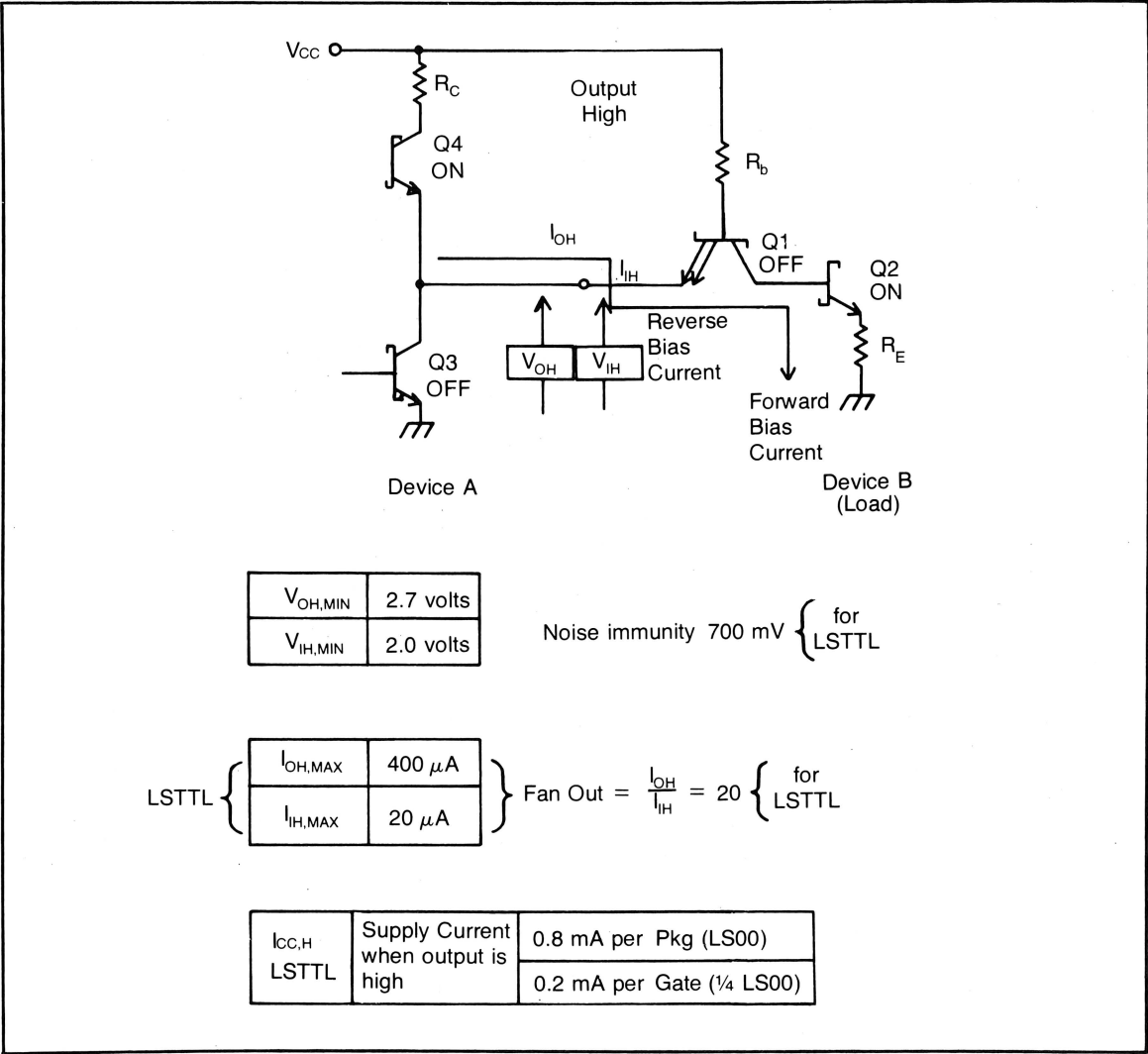


Fig. 7-7. One TTL device in output high state driving another TTL input, with key parameters listed.

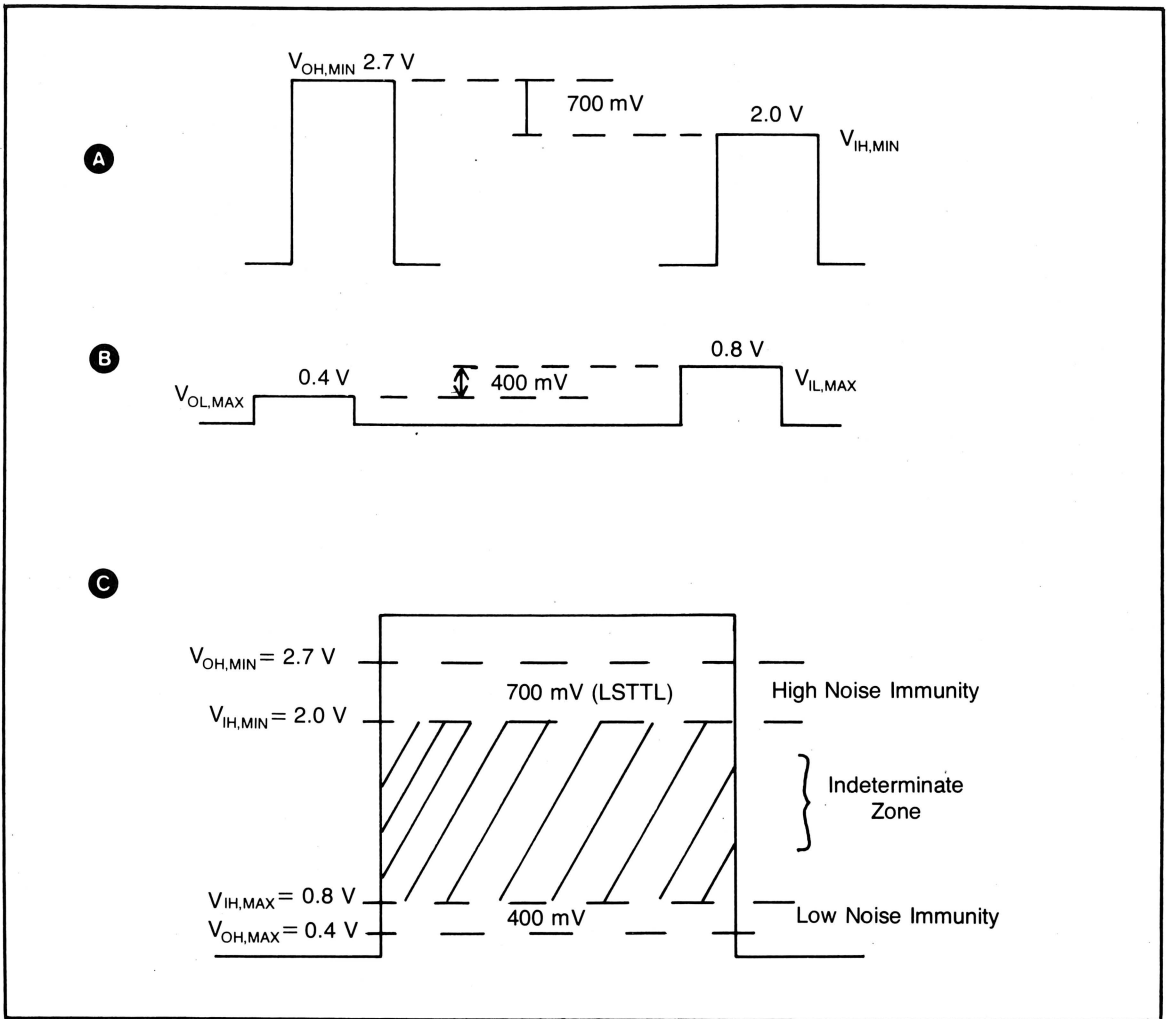


Fig. 7-8. Noise immunity values for a typical LSTTL device for A) logic high and B) logic low. C) An alternative representation illustrating high and low input and output logic levels, as well as noise immunity and the indeterminate zone between allowable input levels.

from V_{CC} through R_c and Q4 of device A (the totem-pole), then through Q1 in the wrong direction and finally through Q2 and R_e . Because the source current is directed in the reverse direction for forward bias of Q1, Q1 is off. However, Q2 is on. (The exact logic function of device B depends on the circuit arrangement downstream as it were, not shown in the figure). Again, referring the parameters already listed ($I_{OH} = 400\text{ }\mu\text{A}$ and $I_{IH} = 20\text{ }\mu\text{A}$ for LSTTL), you can see that the fan-out for the high state is 20. Also, the fan-out for STD-TTL

is $I_{OH}/I_{IH} = 400/40 = 10$. Obviously, it is no coincidence that the fan-outs for high and low output states are the same for any subfamily.

Regarding supply current for output high, LSTTL current consumption in the high output state is 0.8 per package and 0.2 mA per gate. For STD-TTL, the values are 4.0 and 1.0 mA. Again, LSTTL uses 1/5 of the power of a standard device.

Finally, as you can see by comparing the values of I_{CC} for the high state and I_{CC} for the low state, the supply current requirements of any TTL device

are less in the high than in the low output state—1/3 to be exact.

Dynamic Parameters

Each major subfamily of TTL has a characteristic range of both *speed* and *power consumption*. The speed of a device can be defined in a number of ways, but the simplest definition is the time it takes a signal to propagate from the input to the output of a device. Power can be defined in actual units of watts or milliwatts per gate or per package. Often, power consumption is specified in terms of the current that the device draws. Finally, *noise* is defined as any undesirable signal propagated through a circuit. In digital devices, noise is a result of the rapid switching between on and off states and manifests itself as spikes or transients.

As a general rule, a fast device—one which can operate at high frequency and therefore handle more data per unit time—tends to consume more power and also to generate more noise. This is a fact of life for both the designers and users of digital ICs. Choosing the right device for a particular application always involves a consideration of these dynamic parameters, and a decision as to what is a satisfactory compromise between them. Of course, there are exotic solutions to the speed/power/noise triangle. New subfamilies of both TTL and CMOS and even hybrids of CMOS and TTL are announced with increasing frequency in the trade publications. But even with these new designs there is an additional tradeoff, one that shouldn't surprise anyone: cost!

Regardless of your intended application, exotic or not, you should be conversant with these dynamic aspects of TTL. Let's begin with the speed of device operation.

Speed. In Fig. 7-9A we have two simple TTL devices, a noninverting and an inverting buffer. Most digital signals are in the form of square waves, and these are shown on the input and output lines of each device. In order to answer the question as to how long it takes such a signal to propagate through such a device, we have to take a closer look at the anatomy of a square wave. This is given in Fig. 7-9B.

Every square wave can be characterized by its height, width, rate of rise and rate of fall. In digital systems the height is prescribed by the demands of valid logic high and low levels. The width is quite variable, and is signified by the term *pulse-width*, t_w . The rise time of the *leading-edge* is also called the *transition* time from low to high, or t_{TLH} . The fall time of the *trailing-edge* of the wave is sometimes called the transition time from high to low, or t_{THL} . In LSTTL devices, the rise and fall times differ slightly, but are usually on the order of 3 nanoseconds (nsec.), that is, 3 billionths of a second.

In Fig. 7-9C, a typical square wave is depicted as it might appear on the input lines of the two buffers shown in Fig. 7-9A. Below this waveform are shown the in-phase output wave for the noninverting buffer, and the inverted output wave for the inverting buffer. As you can see, there is a delay between the leading edge, and its appearance at the output, called the *propagation time* for the low to high transition— t_{PLH} . This time is measured between two comparable points on the leading edge, and indicated by the dashed lines for both the inverted and noninverted outputs. Also indicated is the *propagation time* for the high to low transition— t_{PHL} .

It is evident from the figure that the speed of the device is defined in terms of the propagation delay for the leading and trailing edges individually not for the square wave signal as a whole. Propagation delay is essentially the result of the internal, distributed capacitance within the integrated circuits. Much of it derives from the junctional capacitance we discussed previously. This small (pico- or trillionth farad) range of capacitance is sufficient to cause a tiny but measurable delay between the time when a transistor (fabricated on the chip) switches from on to off and vice versa. Therefore, we can say that the propagation delay of the edge of a square wave is really the sum of the individual rise and fall times (unijunctional charging and discharging times) of each transistor.

While the two propagation delays (t_{PHL} and t_{PLH}) may be somewhat different, we can use the longer of them as a reasonable measure of overall

propagation time for a signal. In the case of a simple buffer or gate in both STD-TTL and LSTTL sub-families, this propagation delay is about 10 nanoseconds, give or take one or two nsec.

Just how fast can TTL devices operate, then?

The answer is qualified by a number of factors.

First, the total length of a pulse would include both rise and fall times, plus some reasonable minimum pulse width to allow for any oscillations (switching transients) to settle down. Add to this

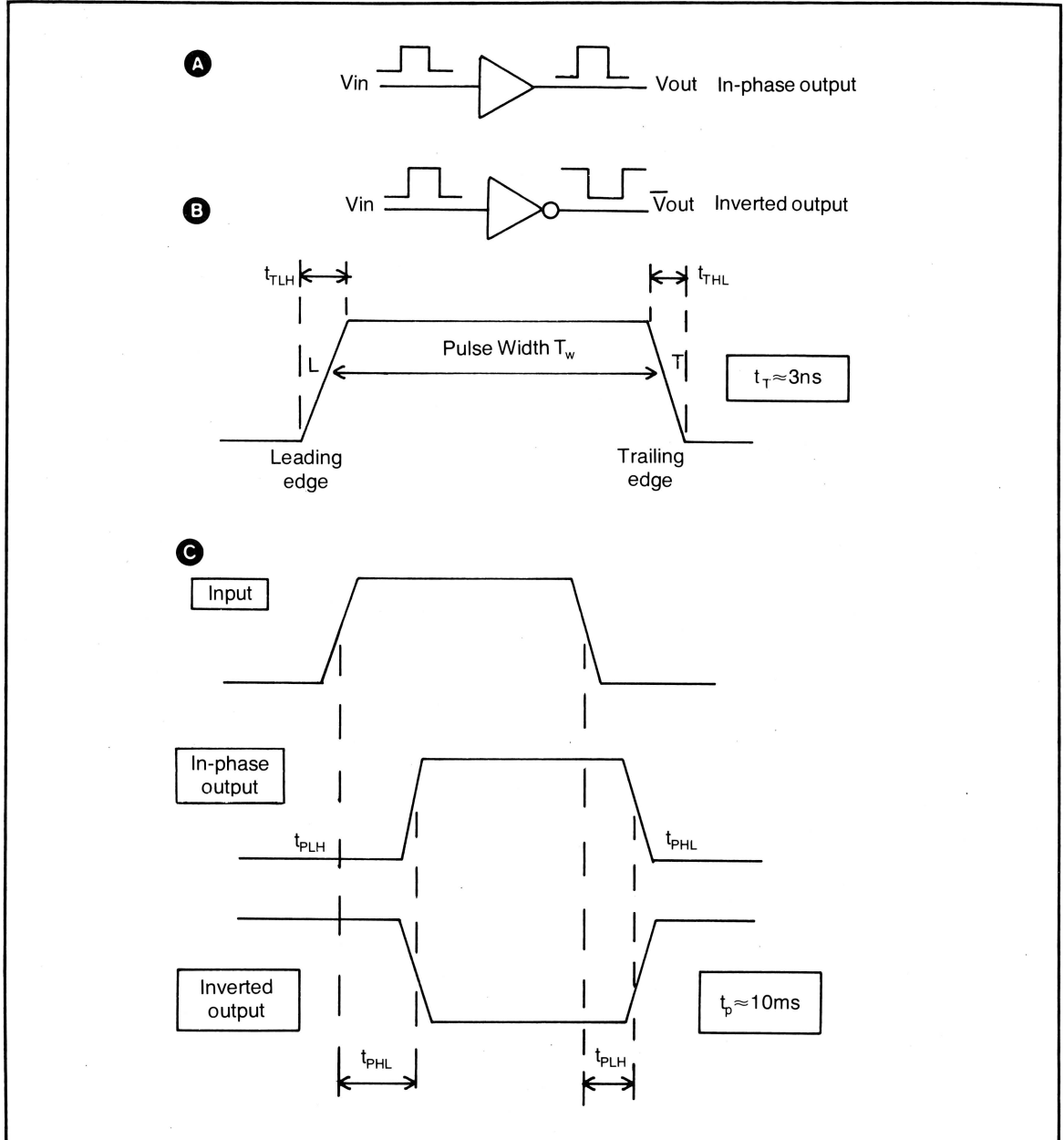


Fig. 7-9. Speed is defined in terms of the propagation delay of leading and trailing edges of a square wave. See text.

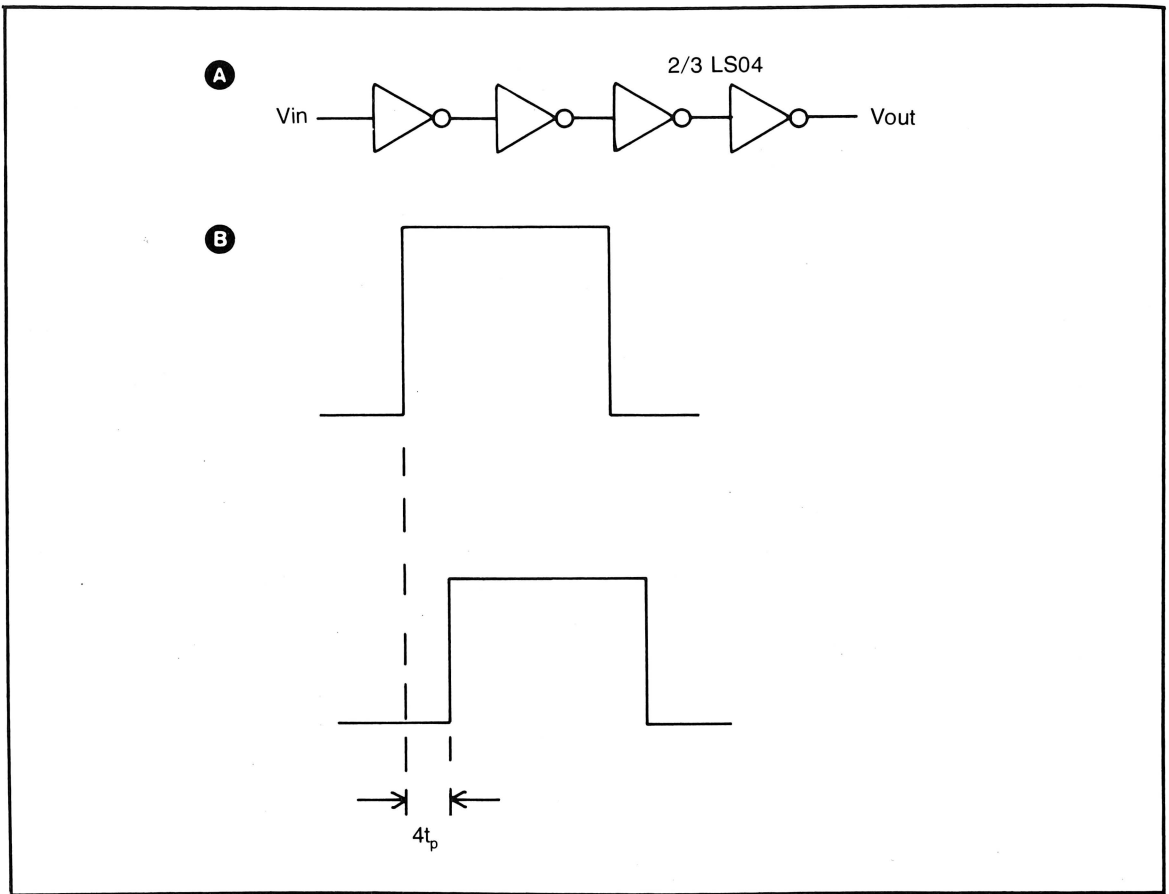


Fig. 7-10. A delay line can be created by placing an even number of inverters in series (A), in order to delay a signal by a small time interval, measured in terms of the same multiple of net propagation delay for the device (B). An even number of inverters leaves the polarity of the signal unchanged.

the propagation delay itself, and you have a rough estimate for a limit on operating speed. Without getting too precise, an LSTTL SSI gate could certainly operate safely at a frequency of 50 MHz or so. This is unrealistic in a practical setting, however.

A working digital system consists of many devices, at all levels of integration. MSI and LSI devices that would exist in such systems have longer transition and propagation times because of their much greater complexity when compared with SSI gates and buffers. Also, digital systems are usually organized into modules or subsystems which must work together. Each such module has its own aggregate *set-up time*, which is the time it takes a signal to propagate through the module and

for the noise transients to settle down. These modules also must be synchronized, and more time is required for that.

Obviously, the upper limit on operating speed is going to be much less than the cruising speed of a single gate in that system. As an approximation, the operating frequency of an LSTTL system of intermediate size might be on the order of 10 to 15 MHz or so, allowing for a reasonable safety margin. Higher speed operation is certainly possible with better design techniques and especially with the availability of new subfamilies of digital ICs. Again, you do pay the piper for such gains with increased cost and power consumption.

As you refer the data manuals, you will see

speed characterized in terms of the parameters just discussed.

With all this talk of propagation delay being a liability, you might wonder if it ever has any practical use. Occasionally, when breadboarding or modifying an existing system, you may encounter a need to delay a signal because of timing problems in your circuit or for other reasons. This need is sometimes met by the simple expedient of inserting a do-nothing device in the system. It simply delays a signal without otherwise modifying it. In Fig. 7-10A, such a circuit is shown. It consists of four inverting buffers in series. The input and output to this series *delay line* circuit is shown in Fig. 7-10B. The delay between input and output is 4 propagation delays, which would be about 40 nsec. if LS04 devices were used.

Set-Up Time. Related to propagation delay, this is the time it takes for a digital IC to be enabled or disabled by some control signal. The term is usually applied to the more complex MSI and LSI devices. For instance, an MSI register may have to be enabled a short time before it can receive and store several bits of data. An enable line or pin may be either active high or active low. In either case, the enabling signal must propagate through the device package before the data to be input will be registered. In essence, this set-up time is basically a propagation delay for the circuitry on the chip that is connected to that pin. (Note that the delay line just mentioned might be employed in roughed-out prototype or breadboarded circuits to provide the necessary pause between the enable signal and data).

Power Consumption and Noise Generation. In any given family of digital ICs, one way to increase speed is to design the devices to operate at higher currents. This is done by lowering the value of the resistors in the logic circuit. You can see this for yourself if you refer to any TTL data book. With higher operating currents, the distributed capacitances (which reside mainly in the transistor junctions and between traces on the chip) can be charged and discharged more quickly. Thus the transistors will turn on and off faster, and effective operating speed is increased. Rise and fall times are

reduced, as is propagation delay.

Of course, if the tradeoff for more speed entailed only more current drain or power consumption, it would be quite acceptable. But greater speed also entails more *noise generation*. The reason for this is understood in terms of distributed capacitance again!

Remember, if you rapidly discharge a capacitor—such as through a low resistance—the resultant current flow can be significant, if rather short-lived. Likewise, when you rapidly charge a capacitor, current will surge to high values. This is a property of capacitive reactance; sudden changes in applied potential (high dV/dt) cause current to pass through a capacitor as if it were a very low value resistance. Such sudden changes in voltage, as occur on the edges of square waves, give rise to the familiar phenomenon of current spikes. They are brief oscillations in a circuit that are also referred to as ringing.

These transients—voltage and current spikes—appear on the power supply lines in a digital system. This is shown in Fig. 7-11A and B, in which a device switches from low to high and then from high to low. As illustrated in Fig. 7-11A, I_{cc} output low is greater than I_{cc} output high as mentioned before. When the output goes from low to high at time t_1 , the current transient is propagated through the system, including the power supply. When the output shifts from high to low at time t_2 , another transient oscillation is generated.

(The switching transient of a low to high transition is somewhat less than from a high to low transition because there are relatively more charge carriers in the output low (current sinking) state than in the output high (sourcing) state.)

At times t_1 and t_2 , in Fig. 7-11B voltage spikes also appear. Again the low to high voltage transients are slightly greater in amplitude and duration than the high to low transients. This *switching noise*, measured as small voltage oscillations, is one of the chief causes of noise in digital systems. This is why a few hundred mV of noise immunity is essential. The noise immunity designed into the devices is not always sufficient alone to avoid problems, however. Other measures—proper de-

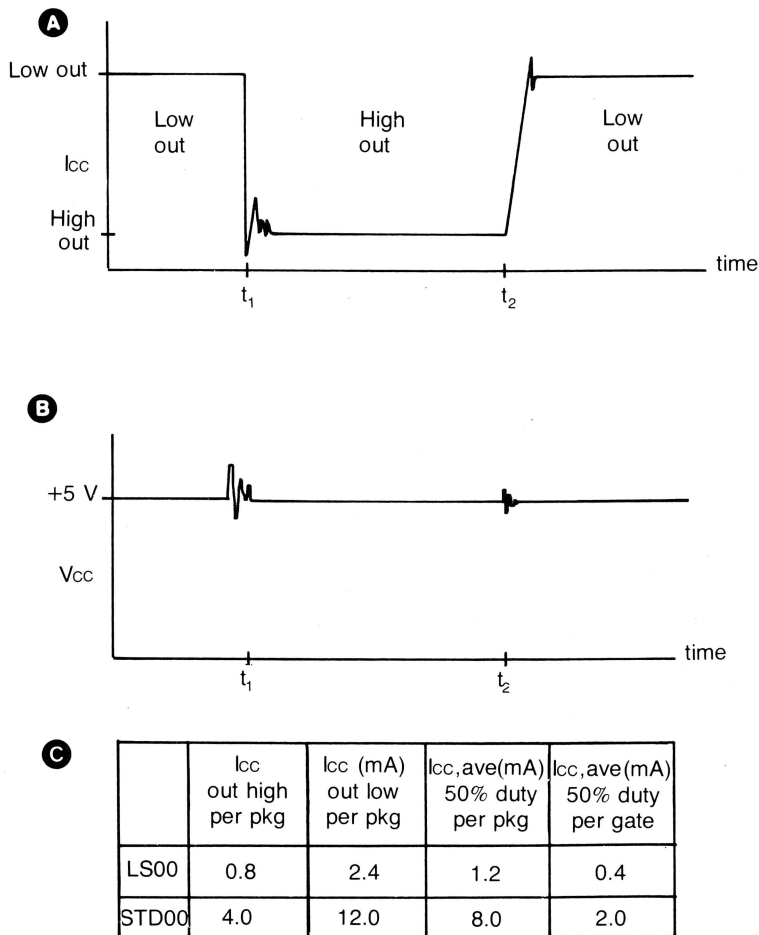


Fig. 7-11. Switching transients create noise in the form of both (A) current spikes and (B) voltage spikes on the supply lines. Note that power consumption is greater for an output low state. (C) Current consumption for a STD-TTL and LSTTL NAND device and package for high and low output, and for a 50% duty cycle.

sign of the power supply itself, proper distribution of power and ground lines in the system, adequate ground, and especially the use of despiking capacitors—are all employed to minimize the effects of noise in a system.

Finally, the current requirements of STD-TTL and LSTTL are restated in the Fig. 7-11C. Per gate and per package values are given. Note that figures for a 50% duty cycle is included. This refers to a condition when the output of the device in question (an '00 or LS00 NAND here) is half the time low and

half the time high. This would approximate average operating conditions.

Summary and Practical Rules

The material presented so far has been fairly detailed, with many of the concepts interrelated. As an aid, the key parameters are summarized in Table 7-1 with values for STD-TTL and LSTTL listed. In brief, LSTTL has about the same speed, yet uses about 1/5 the power and has twice the fan-out rating of STD-TTL. Now you can see why LSTTL has all

but replaced STD-TTL. You can expect that other families of digital ICs with even better speed/power parameters will come along and will replace LSTTL itself.

It is important that you be able to use all this information in a practical sense. To this end, a number of rules for using TTL devices are stated below.

Estimating Power Consumption. Once a circuit is designed it must be powered. Power supplies are rated both in terms of available current, as well as in wattage.

For IC work, the current rating is often more convenient to use because the data sheets specify power needs in terms of ICC per gate and per package. Power dissipation is also specified for MSI and LSI devices, in terms of milliwatts per package, but we'll stick with ICC for convenience. Calculation of

the necessary supply current involves two steps: estimating the current consumption of all of the IC packages in the circuit, and adding the current consumption of any other components or devices other than ICs.

As a rule, you can use the average supply current per package for a 50% duty cycle, ($I_{CC,ave}$) as the figure for current requirement. With many MSI devices, I_{CC} is given for some specific input and loading conditions, as you can see by referring to a data manual. You can use this as an equivalent of average current need for such devices.

As an example, assume you are using three LSTTL NAND packages, and are driving four LEDs. Each NAND package will consume an average of 1.6 mA. Each LED has a resistor which limits current flow to 8 mA. Then, total supply current should be at least $(3 \times 1.6 + 4 \times 10)$ or 44.8 mA. As a

Table 7-1. Standard and Low Power Schottky Device Electrical Characteristics. (Electrical Black Box Terminal Characteristics.)

	STD-TTL 7400	LSTTL 74LS00	Units
Power			
V_{CC}	5.0 \pm 0.5	5.0 \pm 0.5	volts
$I_{CC, ave}$ gate	2.0	0.4	mA
pkg	8.0	1.6	mA
Output			
V_{OL}	0.4	0.4	volts
V_{OH}	2.4	2.7	volts
I_{OL} (sink)	16.0	8.0	mA
I_{OH} (source)	0.4	0.4	mA
Input			
V_{IL}	0.8	0.8	volts
V_{IH}	2.0	2.0	volts
I_{IL}	40	20	μ A
I_{IH}	1.6	0.4	μ A
Noise Immunity			
Low	0.4	0.4	volts
High	0.4	0.7	volts
Fan-out	10	20	
Speed			
t_{PLH}	11	9	nanosec
t_{PHL}	7	10	nanosec

(Note: The values for propagation delays and $I_{CC, ave}$ may differ somewhat from one manufacturer to another. Values used are from Texas Instruments' TTL Data Book).

general policy, estimate your current needs and then add at least a 25% safety margin, or more. The +5 volts game port supply (pin 1) would be adequate for the requirements, as it can supply up to 100 mA of current, according to Apple.

The limit on the +5 V line on the game port of 100 mA is a nominal value. Apple's +5 V line from the main power supply has a limit of 2.5 amperes. This is the limit for total current to the mother board (RAM, ROM, CPU) and to all peripheral devices that derive their power directly from the Apple. Specifically, the mother board requires about 1.6 amps, and total current drain by all peripheral boards should not exceed 500 mA, according to the Reference Manual. Now allow 100 mA for the game port +5 volt line. If you actually used the 500 mA off the I/O slots plus the 100 mA of the game port, total current drain would be $1.6 + 0.5 + 0.1$ or 2.2 amperes. This leaves only a 300 mA safety margin.

By this analysis, you could draw more than 100 mA from the game port supply with certain limits. Be sure you do not have any peripherals drawing excessive current when you are performing experiments with the game port. For instance, printer cards, disk controller cards, internal modems, and similar peripheral cards typically draw from 200 to 300 mA. None of these are on simultaneously in the typical case. So you do have perhaps 500 mA of available current to use. Naturally, you do not want to use all of it. Thus, a reasonable caution would be to base your requirements on a 200 mA figure and no more. This means then you could calculate a maximum actual current requirement for a particular circuit of, say, 130 mA, and still have a 50% safety margin.

Unused Inputs, Pull-ups and Pull-downs.

It has already been mentioned, but it bears repeating. When you want to be absolutely sure that a device will actually work in a circuit, do not leave inputs floating or unconnected. Specifically, you may want to purposefully set one or more inputs to a gate that you are using to a logic high. You know that an unconnected input sees a logic high, because it is unbiased; therefore, leaving it unconnected may seem alright if you want a HIGH on that input.

Usually this is true, but radiated circuit noise (in larger systems) may be picked up by this floating input and momentarily set it low. Therefore, to be sure of a valid high on that pin, use a 1 to 2 kohm *pull-up* resistor to tie the input high.

In a different situation, if you are using only one or two gates on a four-gate (quad) chip, then ideally you should make sure that the outputs of the unused gates are high. For example, if you are using a NAND package, tie one of the inputs to ground; if using NOR, tie both to ground. This assures an output high in either case, and results in a lower power consumption for the package as a whole. Make sure you use a current limiting resistor of 1 to 2 kohms between the input and ground. This is called a *pull-down* resistor, and is a precaution against your accidentally connecting a high to one of these inputs.

Grounding. The first rule is that your Apple should be connected to a grounded three-pin ac outlet with a true earth ground. (With the computer off but plugged in, you can connect an ohmmeter between the power supply chassis and a water pipe to verify this).

Also, make sure that you have a common ground in all your computer I/O experiments and applications. That is, the Apple's ground (pin 8 on the game connector, or pin 26 on a peripheral card slot) should be common to any external circuitry you may interface with the computer. Sometimes the electrical ground of a circuit is not at earth ground, but at some small potential above or below it. The most common example is that of using a separate power supply to drive your applications circuit. By connecting the external power supply's ground line to the Apple's ground, you can be sure that both circuit and computer ground are common and at the same potential.

Loading Rules. Loading means several things: fan-out, sink current drive capability for devices of a different subfamily, and non-IC devices.

You can drive 20 LSTTL inputs from one LSTTL output, and 10 STD-TTL inputs from one STD-TTL output. Enough has been said about calculating drive capability for devices of different

subfamilies. For discrete devices such as LEDs, small speakers, etc., you must not sink more current than your device can supply. Current limiting resistors are used when necessary. If you need more power, transistors or buffer-drivers (covered next) are employed.

Despiking. Despiking capacitors are used to short high frequency noise transients to ground and thereby preventing or minimizing their propagation through the circuit. One uses high grade tantalum capacitors of .01 to 0.1 μF with a 15 to 25 volt rating. One of these capacitors for every 5 or 10 ICs is a suggested ratio. These are connected between the power and ground lines and are distributed throughout the circuit.

TTL AND BUS-ORIENTED COMPUTER LOGIC

Computers are bus-oriented machines. The signals which propagate through a computer circuit are commonly classified as either *data*, as an *address* or as *control signals*. Buses are conduits that distribute signals between the CPU chip (the 6502 microprocessor or central processing unit in the Apple) and the other major components in a typical system—memory and peripherals (disk, printer, video modem, etc.).

In the case of eight-bit microcomputers, the data signals are carried in groups of eight parallel lines with each line representing a single bit of an eight bit word or *byte*. Each line can be high or low, that is, 0 or 1. This permits to 256 different combinations of 0s and 1s to be present on the eight lines. The data itself may be BASIC tokens, machine language commands, numbers, or ASCII characters. Together, this set of eight data lines is referred to as the *DATA BUS*. The data bus is designed to be bidirectional—each line on the data bus can pass signals from the CPU to the memory/peripheral or vice-versa.

Similarly, the address lines (usually 16 in an 8-bit micro) are also physically grouped as a set of parallel lines and distributed throughout the computer as the *address bus*. The address lines access memory locations for storage and retrieval of information. The CPU normally controls the address bus to transfer information to and from memory.

However, special peripherals can also directly access memory without the intervention of the CPU; this is called direct memory access (DMA).

A third group of signals is more diverse in character. It includes clock or oscillator lines for timing and synchronization, a read-write line for control of data direction, a line for system reset, and several other lines for direct memory access and system interruption by peripheral devices. Collectively, this set of lines is often known as the *control bus*. Signals may emanate from either the CPU or from a peripheral, depending on the identity of the line. The number of such lines varies, depending on the 8-bit machine with which you are dealing. Usually there are a dozen or so control lines.

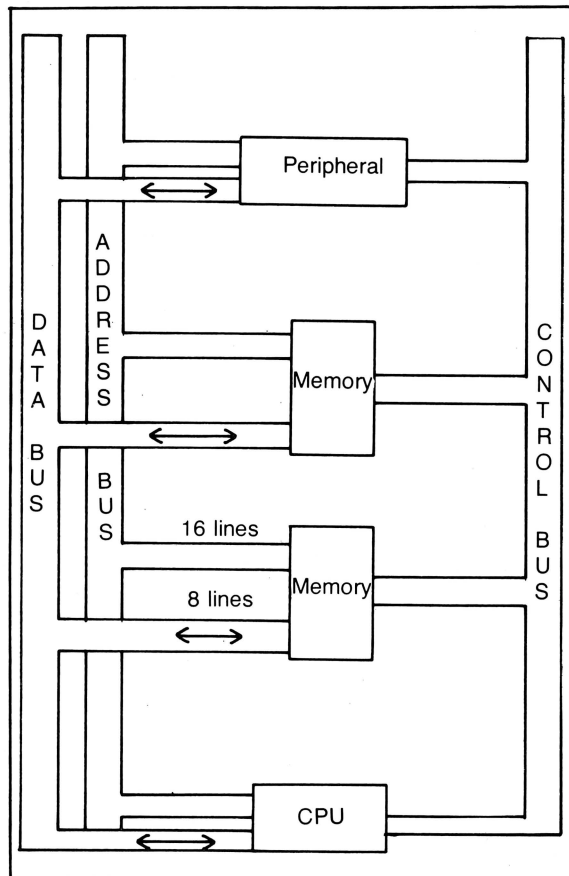


Fig. 7-12. The system bus of a microcomputer includes data, control and address buses. See text.

This simplified scheme is shown in Fig. 7-12 with three main buses deriving from the micro-processor chip (CPU) itself. Machine organization is much more complex, but this gives you the basic idea.

One question arises. Since these three system buses must be *shared* by the host of on- and off-board modules (memory, I/O circuitry, peripherals), just how do you prevent conflict among them? There are literally tens of thousands of gates that must share the same line—remember, you probably have 48 K or more of memory alone, and each memory cell must share the data and memory bus individually!

Three-State Logic

Bus-oriented logic is a very efficient way of distributing signals in complex digital systems. But as you've just seen, many devices must share the same bus lines at different times. The intent here is certainly not to detail computer architecture. But the solution to this problem of shared lines is important nonetheless because it leads us to the concept of *three-state logic*.

The problem of connecting two TTL totem-pole outputs to the same bus line is illustrated in Fig. 7-13. (In reality, many such outputs would be connected to any given line in a data, address, or control bus. Two are shown for simplicity).

Assume that the output of device A is LOW and that of device B is high. Then, Q3 of device A (Q3-A) will sink current through the pull-up transistor of device B (Q4-B). This current we will call the sink current of device A ($I_{A,sink}$). Now $I_{A,sink}$ will be significant—say a milliamp or more—and will certainly exceed the maximum specification for TTL output high (source) current of $400\ \mu A$. Obviously, there may be an indeterminate logic level on the bus line. Also such excessive current drain, if sustained, will destroy pull-up transistor Q4-B!

The solution to this high/low conflict on the bus is to isolate all the outputs from the bus line except the one which is to place a signal on the line. Specifically, if you could somehow turn off both of the output transistors of a device, you would achieve such electrical isolation. In Fig. 7-14, the

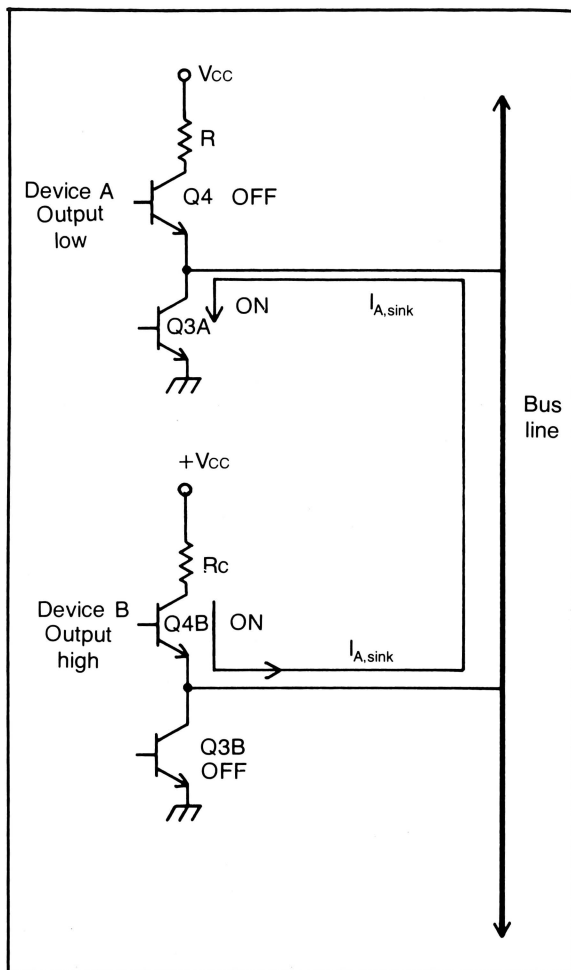
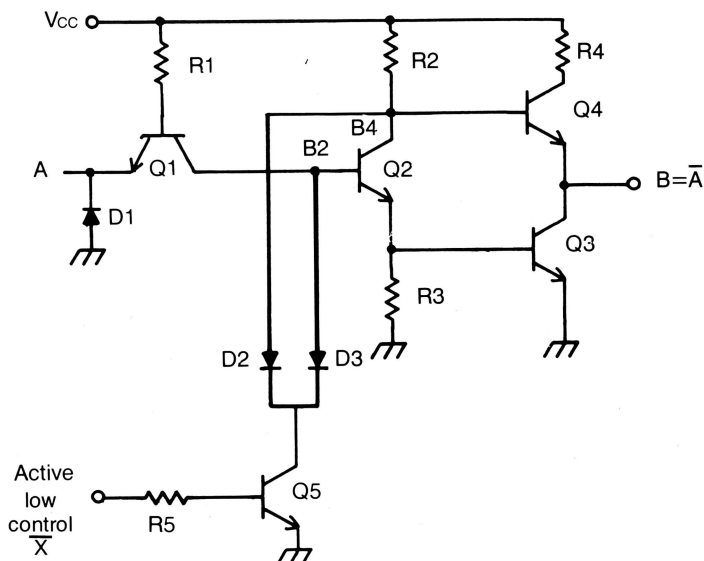


Fig. 7-13. Problems arise when simple totem-pole outputs must share a bus line. See text.

additional circuitry necessary is drawn as thick-lined circuit symbols, and consists of two diodes, one transistor and a resistor. This is the very simplest example of the three-state logic device.

This circuit operates as an inverter. (Analyze the on and off states of each transistor Q1 through Q4 to prove this to yourself). When the control line (X) is low, D1 and D2 are reverse-biased, and the control circuit has no influence on inverter operation; the output is either high/1 or off/0.

When the control line is set high, Q5 saturates (because R5 is selected appropriately), its collector goes to near zero, and both diodes conduct. The



X	Q5	Q3	Q4	State
High	On	Off	Off	Disabled
Low	Off	Depends upon input A		Enabled

Fig. 7-14. Three-stage logic is an ideal solution to bused logic.

bases of both Q2 and Q4 are near zero because of the low-resistance pathway to ground through the diodes and saturated Q5. Q3 and Q4 are off/nonconducting and, therefore, are present as very high resistances. Regardless of the state of input A, both Q3 and Q4 will remain off. That is, the device is effectively *disabled* by a high control signal, and *enabled* by a low control signal, as indicated in the table below the circuit.

Figure 7-15 illustrates what the output of a disabled three-state device looks like; both Q3 and Q4 appear as open circuits, and the output is electrically isolated from any circuit attached to it. This near infinite output resistance is the third state of the three-state device. The three state output for bus-oriented logic was pioneered by National

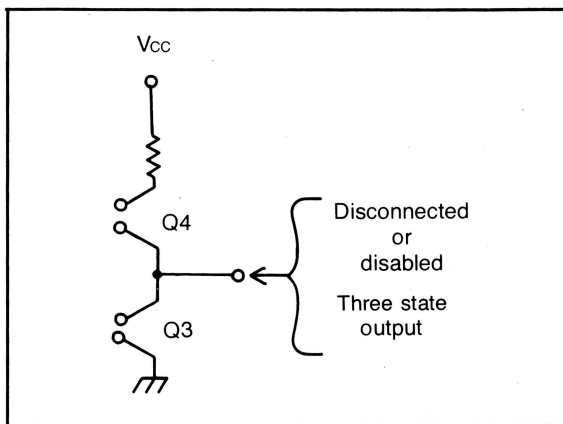


Fig. 7-15. When disabled, a three-state device is effectively removed (electrically) from the bus line or device which it drives.

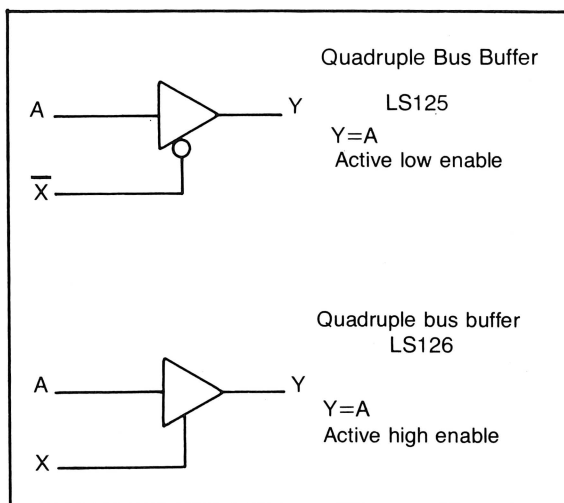


Fig. 7-16. Examples of common three-state buffer-drivers.

Semiconductor, and is also known under the proprietary name TRI-STATE, a trademark of that company.

Three-state devices are in part characterized by their control lines. By convention, an active high control means that the device is enabled by a high signal on the control, and an active low control means that the device is enabled by a logic low.

Of course, the circuitry in existing devices on the market is rather more complex than that shown in Fig. 7-14, but the organization is quite similar.

Figure 7-16 gives examples of three-state buffers that include the 74LS125, a noninverting buffer with an active low enable, and the 74LS126, a noninverting buffer with an active high enable.

A simple application example of three-state logic is given in Fig. 7-17. A1 through A4 represent

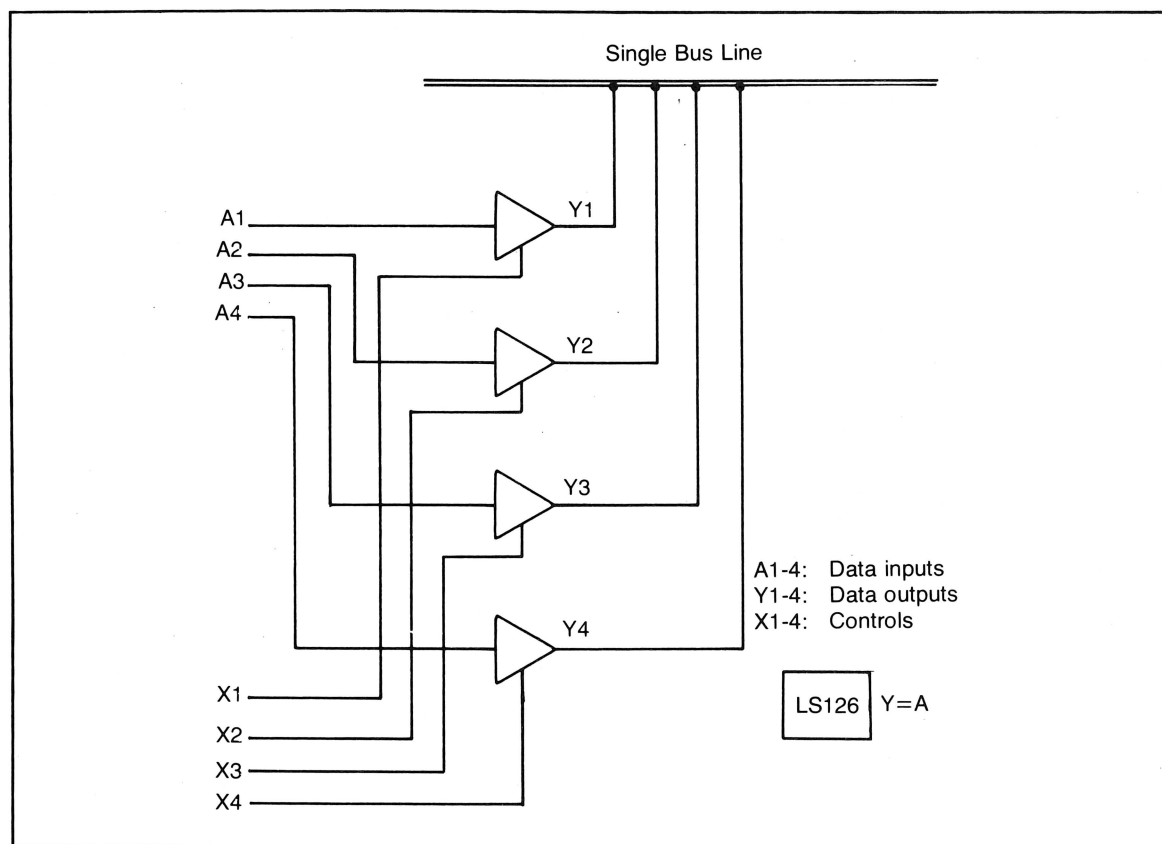


Fig. 7-17. Typical application of three-state buffers, wherein four data sources can share the same bus line. Control signals are usually decoded addresses from the system address bus.

data signals. Only one of these lines should be placed on the signal bus line at any one time. The noninverted data outputs (Y1 through Y4) are connected to this line. X1 through X4 are the active high enable lines that control the output—a high control signal allows for normal device operation, while a low signal disables the device, electrically removing it from the line. Again, real situations are much more complex; in a typical computer system bus a set of parallel bus lines is shared by many sets of three-state output devices.

In general, any digital device can be designed with a three-state output. This includes SSI circuits as well as more complex MSI devices.

There is a second major concept implicit in Fig. 7-17 which should be mentioned briefly. You may wonder as to how the computer designer arranges to have the appropriate set of related three-state devices appropriately enabled or disabled. That is, how do you turn all the devices off except those which are to place signals on the bus? The answer is *address decoding*. The CPU controls the address bus at all times (except in the special case of direct memory access by a peripheral). The address lines on the address bus are first *decoded* by off-the-shelf chips designed for that function. When the correct address is placed on the address by the CPU, these decoded signals are then used to enable the desired devices. This may be done by machine language or even by direct PEEKs and POKEs from BASIC. Address decoding is a topic to be picked up later in Chapters 10 and 11.

Buffer-Drivers.

The LS125 and LS126 devices just mentioned are also known as buffer-drivers. Buffers, as you know, perform the trivial logic operations of NOT or IS. A buffer's main function is really to normalize a signal to a standard strength so that it is at full fan-out or drive capability. Taking this one step further, there is no reason why a given device cannot be designed to provide even greater drive capacity than the other devices in its same subfamily. This is usually in the form of greater sink current. While the device may consume more power, the greater available current is particularly useful in

driving current hungry loads and especially capacitive loads such as transmission lines.

The augmented sink current of these buffer-drivers is several-fold more than standard output devices within the same subfamily. LSTTL devices have an 8 mA sink capability. But an LS buffer-driver, such as the LS125 or LS126, has a sink current rating of twenty-four (24) mA. That is, it is capable of driving up to sixty (60) LSTTL inputs!

By the way, source current capability is also increased. Depending on the device, I_{OH} is about 5.2 mA for a typical quad or hex buffer-driver.

Another means of rating the drive capability of these devices is to specify the nominal value of a resistive load to which they can supply sink current and still maintain a valid output low. For example, you remember that a STD-TTL device has an I_{OL} of 16 mA, and this corresponds a minimum value of about 300 ohms for a load. Now a STD-TTL buffer-driver has an even higher value of I_{OL} than LSTTL drivers, as you would expect. The 74128 has an I_{OL} of 48 mA and is referred to as a 50-ohm *line driver* because its major application is to drive bus and transmission lines.

There is another meaning of the word *buffer*. In this sense of the word, to buffer means “to isolate” a device from the rest of a system. In other words, buffering a signal could imply that you are feeding through it a device with a three-state output. This need arises wherever a signal distribution line is shared within a digital system. By all means, when reading articles or other technical literature, try to make this distinction yourself, even if the author does not do so explicitly.

EXPERIMENT 10, LSTTL SINK CURRENT MEASUREMENT AND AUGMENTATION

Purpose

To measure the sink current rating of a typical LSTTL device, and of a ganged-parallel hex inverter configuration.

Materials

- | | |
|-----------------------|-----------------------|
| 1 - LS04 hex inverter | 1 - 100 ohm resistor |
| 1 - 1 K resistor | 1 - 5 K potentiometer |

Procedure, Part I

1. Assemble the circuit in Fig. 7-18A. Using a voltmeter, measure the output low voltage with the potentiometer at 5 K. Then gradually decrease the pot until the voltage reaches the maximum allowable for a TTL output low state, 0.4 volts.
2. Calculate the output low current at this point by first removing the pot from the circuit, and then measuring its resistance. Use the formula:

$$I_{OL} = (V_{CC} - 0.4) / R_{load}$$

where $R_{load} = R_L + R_p$.

3. Repeat steps 1 and 2 for a number of inverters, either on the same or different IC packages.

Discussion

You should have measured a voltage of about 0.10 to 0.15 volts for the minimally loaded state where total load resistance was 5.1 K. You then measured the value of R_p at $V_{out} = 0.4$ volts, and then calculated the value of the output low current at this voltage. You should have obtained a value in excess of the maximum value of I_{OL} specified for LSTTL. A value of 12 to 14 mA would be typical. R_{load} was probably in the range of 350 ohms or so (much less than the 600 ohm range at the 8 mA rating mentioned earlier). These results suggest that LSTTL devices have significantly greater sink current drive capacity than what is specified in the data sheets. This is why you can get away with using an otherwise unloaded LSTTL output to drive

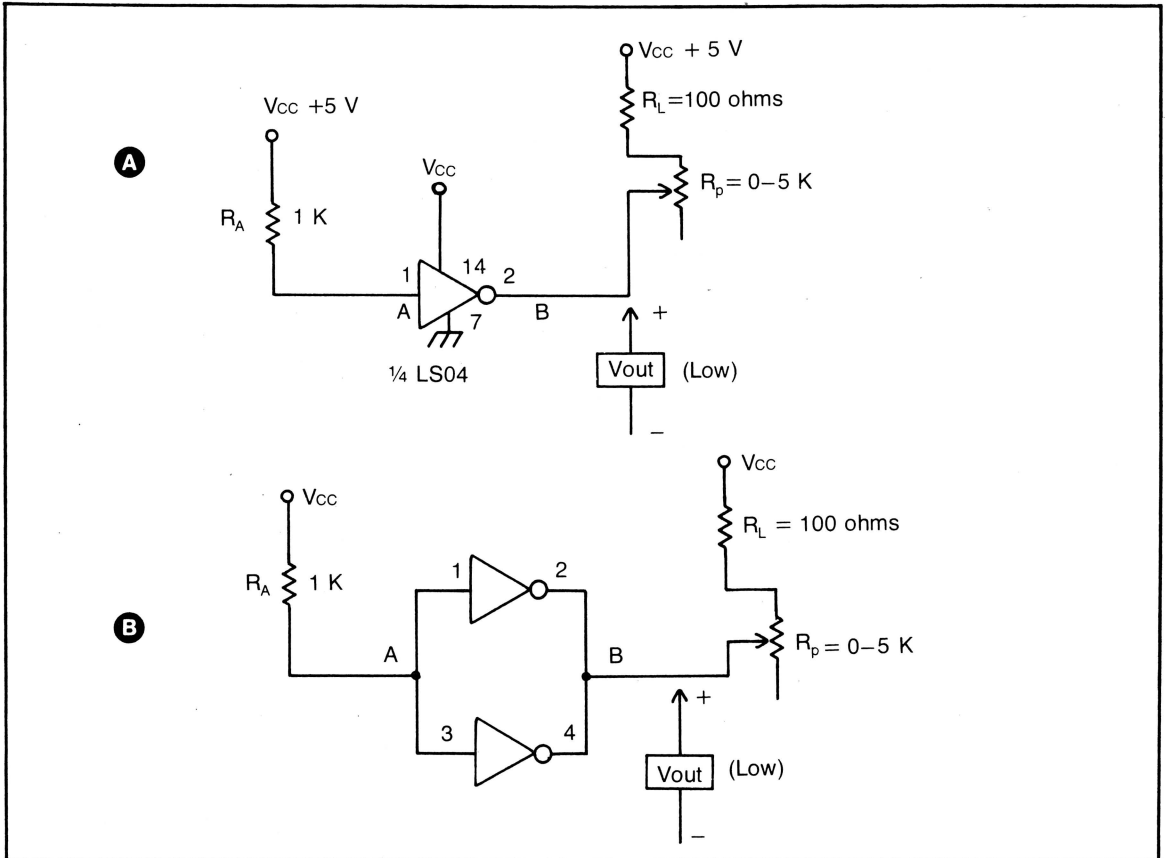


Fig. 7-18. Experiment 10. Experimental circuits for measuring I_{OL} (sink) current drive capability of (A) a single inverter and (B) a ganged parallel inverter.

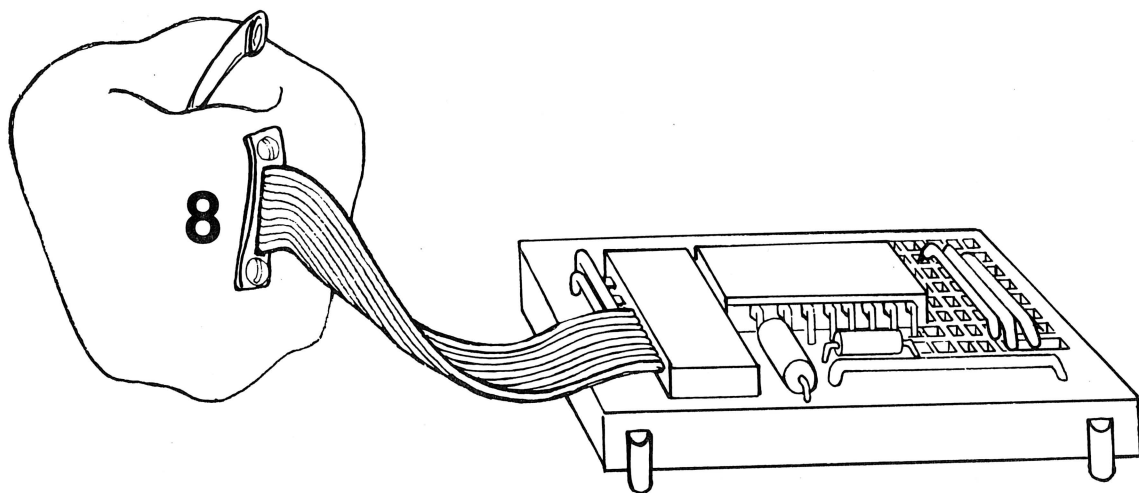
an LED, which would normally require something in the 12 to 16 mA range for adequate brightness levels. Naturally, the same overdesign or safety margin for current drive applies to other TTL sub-families.

Procedure, Part II

1. Build the circuit in Fig. 7-18B.
2. Using the same procedure in steps 1 and 2 above, decrease R_p , measure it at $V_{out}=0.4$ volts, and calculate I_{OL} .

Discussion

This parallel-ganged hex inverter gives you roughly twice the drive power of a single LSTTL output. Values of I_{OL} of around 25 to 28 mA would be typical. The same ganged configuration works in STD-TTL as well. If you wanted to triple output drive, then you could put three such devices in parallel. (Decrease the value of R_L to 47 ohms if you want to make the actual measurements).



SSI Sequential Devices

Digital devices can be divided into the two broad categories of combinational and sequential logic. The building blocks of all digital devices are the SSI combinational devices which you studied in the early chapters. Now we'll turn to sequential logic, again keeping within the bounds of small scale complexity (12 or fewer gate equivalents per IC device).

In this chapter, you will learn about flip-flops and one shots. Logic schematics, timing diagrams, truth tables (called state tables) and verbal descriptions will all play a role in the discussion of sequential logic. The emphasis will be on key terms, on the basis SSI sequential functions, and on experiments which demonstrate the operation of typical device packages. Simple applications for these devices will also be illustrated.

TYPES OF SEQUENTIAL DEVICES

The fundamental difference between sequential devices and combinational devices is the use of

one or more feedback loops from the output back to the input. This feedback allows these devices to retain a prior state and compare it with current and future inputs. This is unlike the case of combinational devices in which prior states have no influence on the current state of the device. Sequential devices have a form of *memory*, and they can generate a *sequence* of output states. Sequential devices provide the basis for a number of digital components—from the simple SSI flip-flop with only two states, to MSI counters and registers. At a somewhat higher level of integration, (LSI and VLSI to ULSI) sequential and combinational functions are merged. Registers, arithmetic units, counters, and control logic all combine to form such components as microprocessor chips, peripheral controllers and computer core memory (RAM).

Basically there are three major types of devices that may be placed in the sequential SSI category. Let's take a brief look at them.

- Bistable devices—flip-flops or latches.

- ☐ Monostable devices—one shots or timers.
- ☐ Astable devices—oscillators or clocks.

Bistable devices may assume either of two states. The outputs of such bistable *multivibrators*, as they are sometimes called, will persist unchanged until the inputs change. That is, these devices will remember their current state indefinitely until a new signal comes along. When the output of such a device is high/logic 1, it is said to be *set*. When the output is low/logic 0, it is said to be reset. Sometimes there are two complementary outputs, and the device is said to have *dual rail* outputs.

Because bistable components can be placed into either of two stable states, given the appropriate inputs, they may be thought of as flipping between these states, much like a flipped coin. Hence the term *flip-flop*. Because flip-flops can *latch* onto or retain a state, they are the basis for a host of devices, including RAM memory, counters, sequence generators and dynamic storage registers.

Monostable devices, on the other hand, have only one stable state. By placing a short pulse on the *trigger* input of such a device, the output will momentarily change to the complementary state, and then revert back to the stable state. The duration of the temporary state is determined by external components, usually an RC time constant. Because of this, and because of their internal circuitry, many integrated astables are not really digital devices at all, but rather linear devices. Monostables will provide a logic level (either high or low, depending on the device) of set duration. This single output pulse is useful to many applications, particularly those involving timing. Monostable multivibrators are often referred to as *one shots* or *timers*.

Astable devices have no stable state. They vary continuously between logic high and logic low output states. Actually, they are nothing more than square wave oscillation. This is why astables are often referred to as *clocks*. The frequency of oscillation is, like the one shots, determined by external RC components. Therefore, they also are not pure digital devices. An example of an astable device is

the hex inverter clock of Fig. 5-25.

Our main concern in this chapter is with flip-flops. However, some attention will be given to one shots at the end of the chapter. The first device we'll examine will be the simplest flip-flop, the R/S latch.

R/S FLIP-FLOPS

Remember that virtually all digital devices can, theoretically, be constructed using only NAND elements by means of the NAND/NAND logic approach. This includes sequential devices too. With only two '00 NAND gates, we can construct the simplest flip-flop, the R/S latch.

The Nongated R/S Flip-Flop

The ungated R/S latch has two inputs, set and reset, as indicated in the Fig. 8-1A through F. These are active low inputs, and are represented by R' and S'. There are also two complementary outputs, Q and Q', one from each NAND output. One may think of these as set and reset outputs: when output Q is high/1, the latch is said to be set; when reset output Q' is high/1, the latch is said to be reset. These outputs are cross-connected to the unused input of the opposite gate, as shown.

Operation of the latch is as follows. First you power up the device. It may assume either output state—set (Q is high and Q' is low) or reset (Q is low and Q' is high)—when the power is turned on. In Fig. 8-1A we assume it comes on in the reset state. Both S(et) and R(eset) inputs, indicated by the negated R and S, are at logic high. As long as both R and S inputs are high, the latch will maintain its current reset condition until we change the input.

In Fig. 8-1B, we set the device by briefly grounding S' via switch 1. This forces the output of gate 1 high (Q=1), and the associated input of gate 2 high. With two highs on gate 2, its output will go low, as will the connected gate 1 input. When switch 1 is released, this set output state will persist. S' will revert to high, but with the other input of gate 1 low, Q will still be at logic high/1. This is shown in Fig. 8-1C.

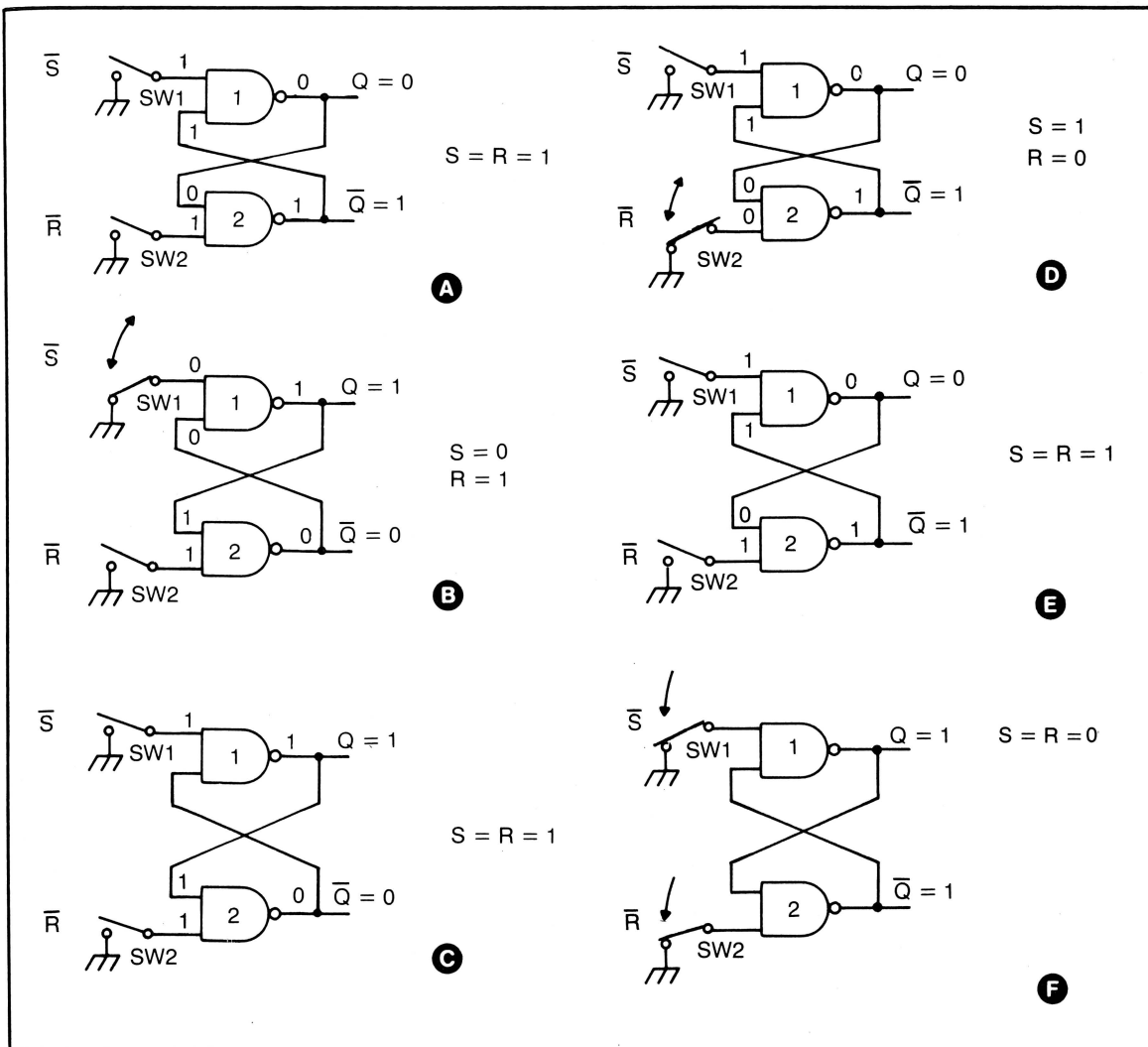


Fig. 8-1. Operation of the ungated R/S latch.

In Fig. 8-1D, the latch is reset by momentarily grounding the R-input of gate 2 through switch 2. This forces the output of gate 2 to logic high/1. Both inputs to gate 1 are now high/1, forcing its output low/0. As a result, the crossed input to gate 2 is also low. When switch 2 is released, as in Fig. 8-1E, the crossed input to gate 2 remains low, because both gate 1 inputs are still high. The reset state persists with $Q=0$ and $Q'=1$. We have come back to the state with which we began, reset.

Finally, in Fig. 8-1F, we have the fourth possi-

ble input state, wherein both R' and S' inputs are low/0. In this case, both Q and Q' will be high. This is an undesirable state for two reasons. First, in normal operation the outputs of a flip-flop will be complementary, never equal. This is because one may want to use both inverted and noninverted levels from the device in practical circuits. A more significant objection to this input condition is that when the inputs return to the high (memory) state, the output will be unpredictable; the device may assume either a set or reset output. Therefore, the

presence of simultaneous lows on the R' and S' lines is illegal, as far as acceptable operation is concerned.

As a further reinforcement of R/S latch operation, we can use the timing diagram and state table in Fig. 8-2.

The timing diagram of 8-2A is keyed, letter for

letter, with the sequence of states illustrated in Fig. 8-1A to F. You can see how the complementary output levels (Q and Q') change as the R' and S' lines are pulsed low. States A, C and E are the memory states, in which the respective outputs of this bistable device remain high or low, as the case may be. In state F, there are simultaneous lows on R' and S',

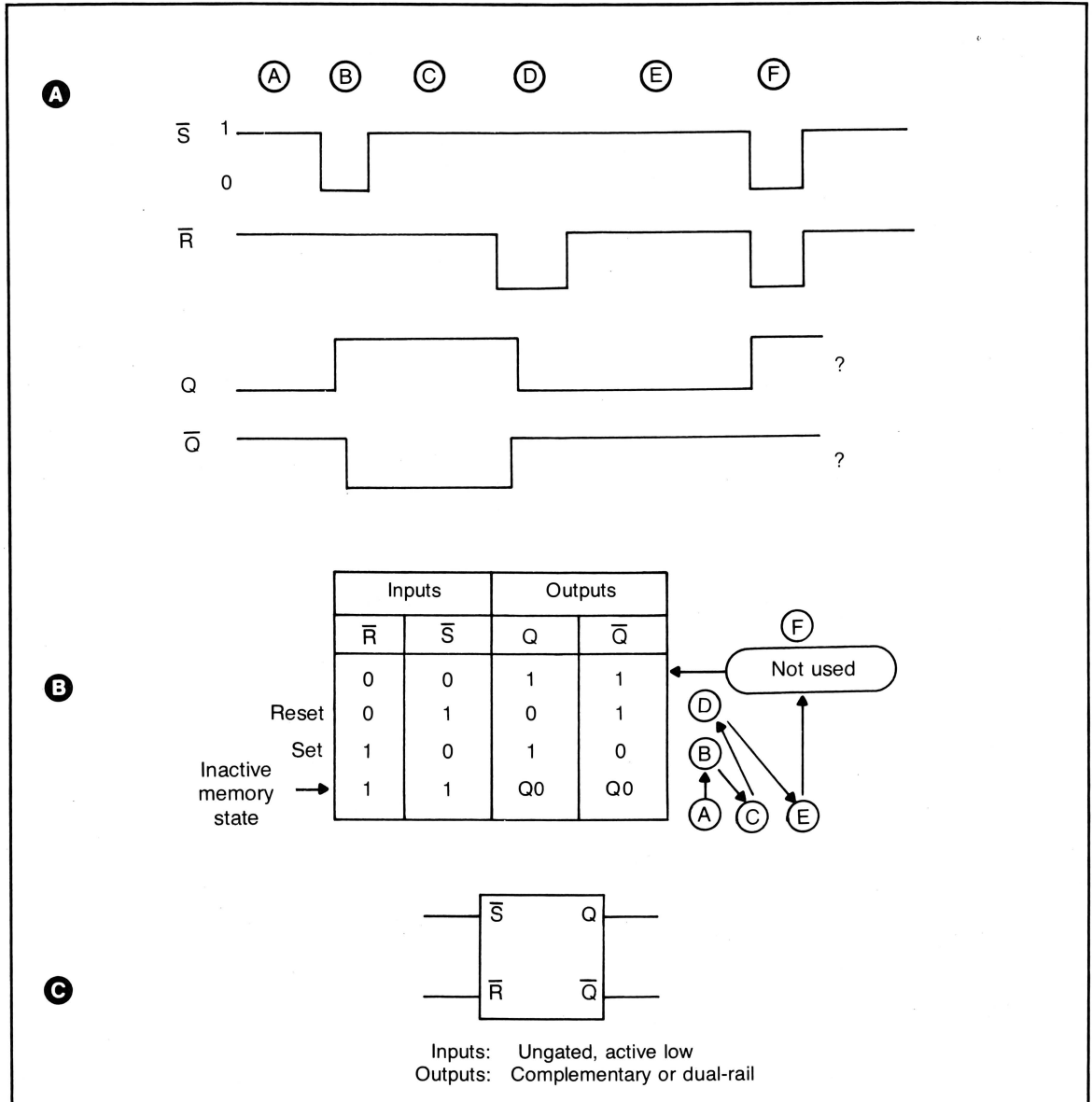


Fig. 8-2. (A and B) Timing diagram and state table for the ungated R/S latch, keyed to Fig. 8-1. (C) Logic symbol.

and both Q and Q' are high. But when both inputs are brought high again, the outcome is questionable, i.e., unpredictable.

An important feature of this basic R/S latch, indicated in the diagram, is that the output transitions occur on the *leading edge* of the input pulse, that is, on the high to low transition of R' and S'. This is a general property of all set and reset inputs.

You'll also note that there is a slight propagation delay, indicated in Fig. 8-2A, between the input pulse and the output transitions. This should always be assumed, but will not always be indicated in the sequential timing diagrams to follow. Such delays can and do cause problems, but certain design solutions do exist which minimize them, particularly the use of synchronous clocking, which is explained below.

In Fig. 8-2B, the various states have been indicated in tabular form. This *state table* is also keyed to the series of states, A through F, of Fig. 8-1. The memory state is signified by the symbols Q0 and Q'0 in the output column. The conditions for set and reset are listed in the two middle rows. The illegal input condition with both inputs low is given in the top row.

Such tables are a means of representing a series of outputs from a sequential device. They are called a state tables rather than a truth tables because the relationship between inputs and outputs is not a logical one, that is, one which follows some simple AND-OR-NOT type relationship as defined in Boolean algebra. State tables can be quite complex, as can their associated equations of state and state diagrams. Such representations are necessary in more advanced treatments of such topics as state analysis and programmed logic. Our representations will be relatively simple.

Figure 8-2C, is the logical symbol for the un-gated R/S latch. Active low inputs are indicated by the negated R and S symbols. Again, this is a complementary output, or dual-rail, device. Output transitions occur on the high or low transition, or leading edge, of the input waveforms.

There are three common and fairly basic applications for the simple R/S flip-flop. Obviously, it can be used as a latching device to retain data. A set

of flip-flops can be ganged together in a group of four, eight or more so that data and memory words might be temporarily stored before being transferred onto parallel bus lines in a computer system. Even single flip-flops are useful for holding data for smaller scale circuits.

Latches may also be ganged in series to perform binary counting operations. Most of the register and counting functions just mentioned have, however, been taken over by MSI devices.

Third, the R/S device can be employed as a *switch debouncer*. A mechanical switch, such as the one shown in Fig. 8-3A, is inherently noisy. When moved from on to off (+5 V to gnd) or vice versa,

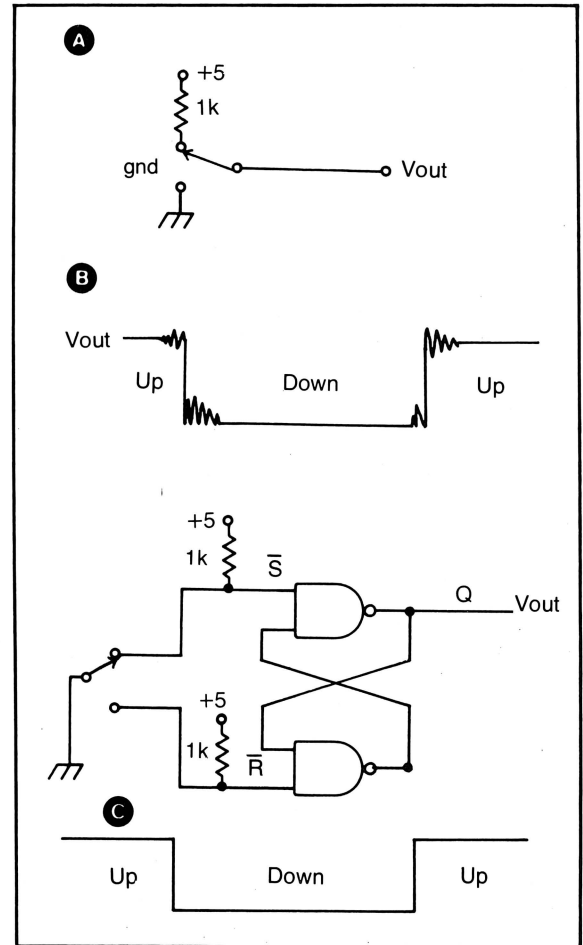


Fig. 8-3. Using an R/S latch to debounce a mechanical switch.

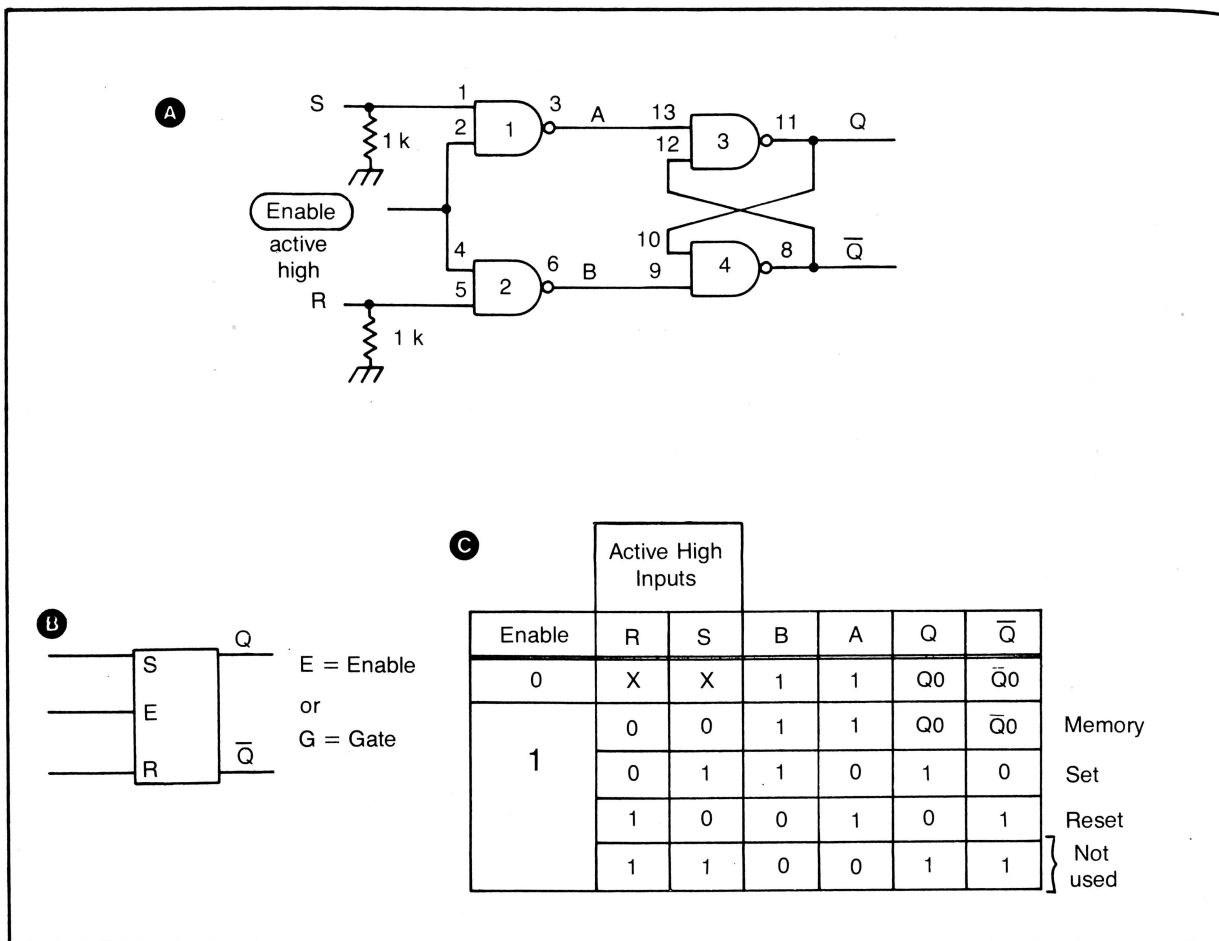


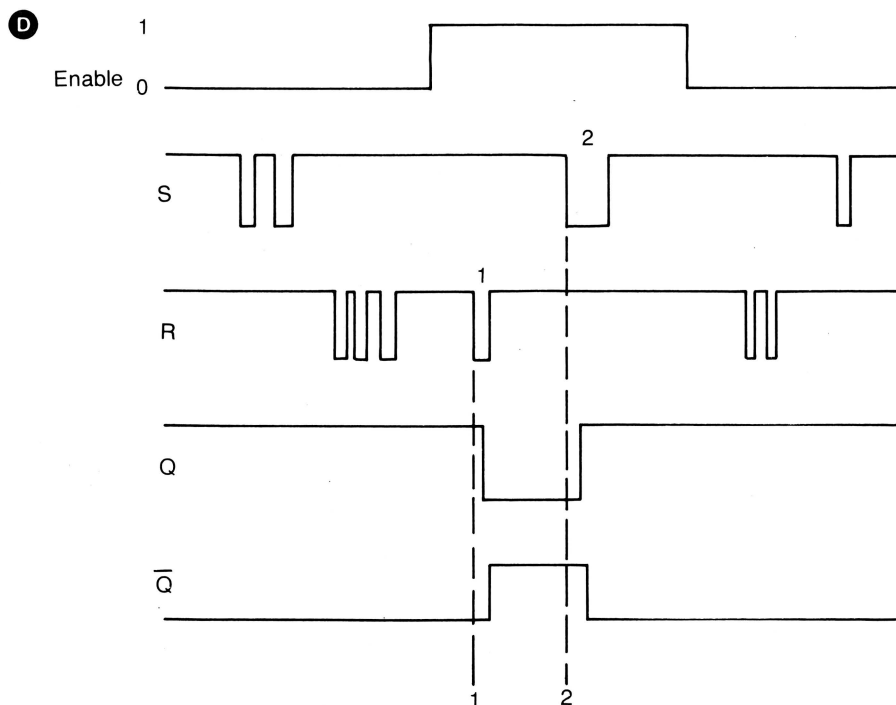
Fig. 8-4. The gated R/S latch. Schematic, logic symbol, state table and timing diagram. The slight propagation delays at edge transitions 1 and 2 in (D) have been exaggerated somewhat.

there is a short interval during which the switch arm bounces against the respective terminal, before settling down. The waveform of this bounce action might look something like that in Fig. 8-3B. Some microscopic arcing may also occur. During this period of intermittent contact, the voltage level may fluctuate and cause spurious signals on the output side of the switch. To avoid this, an R/S latch is interposed between switch and output. Both R' and S' lines are pulled high, with the resistors serving a current limiting role. When either input is grounded, a low or high appears on the single output line. The transitions are sharp, thanks to flip-flop operation; the noise immunity inherent in the de-

vice also prevents false triggering during the mechanical switch transition. The output waveform in Fig. 8-3C is a cleaned-up version of that in Fig. 8-3B.

The Gated R/S Latch

The R/S flip-flop of the last section was always ready to change its output given the appropriate inputs. However, it is usually desirable to *gate* the inputs, so that the device can be enabled or disabled at will. Such an enabling feature enhances the flexibility of the flip-flop, and gives you an extra level of control over the device. A four-NAND circuit which



accomplishes the function of a gated R/S flip-flop is illustrated in Fig. 8-4.

The operation is quite similar to the ungated flip-flop, with a few important differences:

- ☐ The inputs are active high rather than low. They are denoted by R and S.
- ☐ Because of this, the inputs may have to be tied low by pull-down resistors, so that they are inactive unless high signals are placed on them. This is particularly important if mechanical switches are to be connected to these inputs, but not necessary if they are driven by signals from TTL devices.
- ☐ The device will not change its output state

unless the active high enable line is high. This line may be referred in a schematic as E, for enable, or as G, for gate. The logic symbol for such a gated flip-flop is given in Fig. 8-4B.

Operation of this device can be understood by reference to the schematic and the state table in Fig. 8-4A and C, respectively. Remember that the operation follows from the basic NAND function, just as with the ungated latch. The device consists of a control section made up of NAND gates 1 and 2 and a flip-flop section made up of gates 3 and 4.

If enable is low/0, then the outputs of both gates 1 and 2 are high. Inputs to the flip-flop section—A and B in the state table—are both high,

placing it into an inactive or memory state, just as with an ordinary ungated R/S device.

With enable held high, the device is enabled and operates as follows. If both R and S are low, gates 1 and 2 are high, and the device is in a memory state. If S goes high, then the output of gate 1 (line A) goes low, and gate 3s output goes high and sets the flip-flop. If instead R goes high, then the output of gate 2 (line B) goes low, and gate 4s output goes high and resets the flip-flop.

As with the ungated latch, the gated version has an illegal state. But because the inputs are active when high, this undesirable state occurs when both inputs are high, rather than low.

The influence of the enable line is indicated in Fig. 8-4D. Understanding this is important, as it explains certain terminology that you will encounter in the technical literature. When the enable line is low, signals on either the S or R lines have no influence on the output. When the enable line is high, however, the output may change in response to signals on the set and reset lines. Pulses 1 and 2 have an effect of resetting and setting the device, respectively, as shown in the figure; the other pulses have no effect because they do occur on either side of the enabling signal.

Therefore, when enabled, this device is said to be able to pass data freely, or transparently to the output. Another way of expressing it is to say that the device is enabled or triggered by a certain level on this control line, in this case high. As you use the data and applications manuals, you will see certain flip-flop ICs referred to as transparent latches or as level-triggered flip-flops.

The gated latch just presented could be considered as either a transparent latch or level-triggered device. In either case, the meaning of this jargon is now clear: it merely refers to a flip-flop with a gating or enabling line (which may be active high or low) that activates the device.

CLOCKED LOGIC CONCEPTS

Related to the idea of enabling a flip-flop, is the concept of clocking. It is often desirable to have devices and modules in a digital system change state only at specified and predictable times. This

synchronizes the system and avoids or at least minimizes many timing problems. System clock signals are provided by square wave oscillators. If you fed the enable line with a such periodic clock signal, then the flip-flop would likewise be activated only during the regularly recurring periods when the line was high. The generic term for sequential devices which can be enabled is *clocked logic*.

While it is true that virtually any sequential device with an enable line can be clocked, there are a few fine points. Take the generalized flip-flop in Fig. 8-5A, for example. Assume first that this is a transparent or level-triggered device. In that case, the output would change in response to the input during the entirety of the active high clock. This is indicated by the presence of pulses 1, 2, and 3 on the transparent data output line in Fig. 8-5B.

Now assume that this flip-flop is not a level-triggered device. Data on the input line or lines cannot affect the output during the entire high clock level, but only during the transition from low to high of the clock. This type of flip-flop is said to be *edge-triggered*, and is not transparent to data occurring after the low to high transition. In Fig. 8-5B, this is illustrated by the fact that only pulse 1 affects a change in the edge-triggered output (bottom line of Fig. 8-5B), because it falls on the low to high transition. Note that the output changes not just on the edge, but on the leading edge of the clock pulse (LE in the figure). Naturally, the outputs of some flip-flops change on the trailing edge. We'll encounter both shortly.

While the term *clocked logic* is sometimes applied to any flip-flop with an enable or clock line, it should apply only to edge-triggered devices. The enabled or gated flip-flops, the ones that are transparent or level-triggered, are by their very nature not activated at a precise instant in time, as during the brief rise or fall of a square wave. So in fact the term *clock* itself should not apply to these devices. Rather, the term gated or enabled would be more accurate.

Let me introduce one final set of terms: synchronous and asynchronous. *Synchronous* logic is exemplified by those sequential circuits in which all

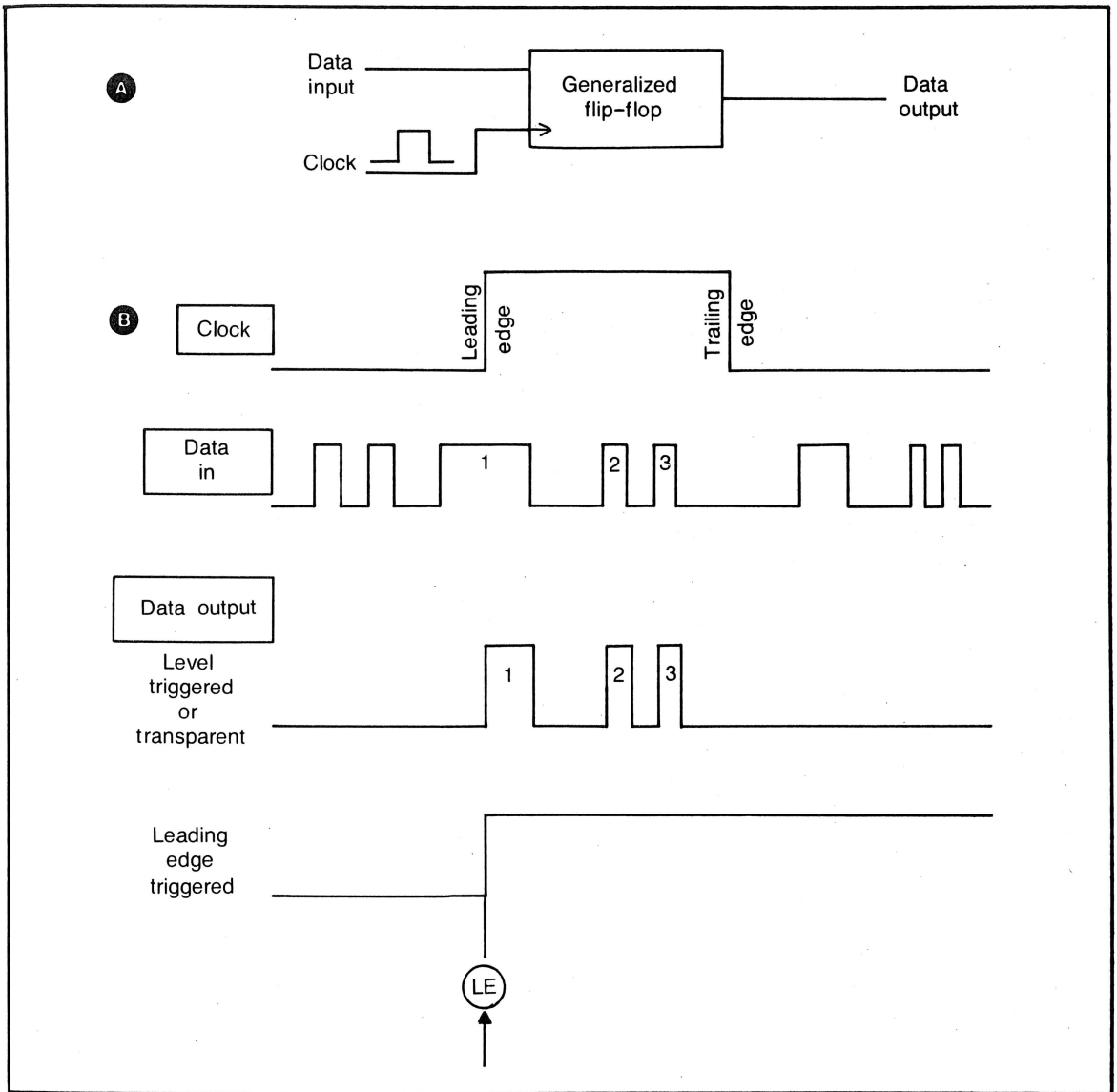


Fig. 8-5. Generalized concept of the level-triggered or transparent versus the edge-triggered flip-flop.

the flip-flop elements change together. This is accomplished having a common clock for all flip-flops. Naturally, these circuits are more complex but also more reliable than *asynchronous* circuits, in which state-changes are not simultaneous. An example of an asynchronous circuit is the ripple counter given at the end of this chapter. Synchronous counters are presented in Chapter 9.

Strictly speaking, the term synchronous logic is synonymous with the term clocked logic. It stands to reason that true synchronization can occur only in a narrow instant in time, not during some broad, poorly defined, interval. Therefore, the use of the term synchronous logic should apply only to edge-triggered sequential devices.

In common usage, these terms are also applied

to input lines. For instance, in the data manuals, an input may be referred to as being synchronous. This means that it can influence the output state only during the edge or transition of the clock signal. An asynchronous input, on the other hand, is totally independent of the clock, and in fact overrides all other inputs. The prime example of an asynchronous input would be the reset or clear line of a counter, which sets all the outputs to zero, irrespective of the state of the clock or of other control lines.

The symbolic representation of level- and edge-triggered lines is given in Fig. 8-6. In Fig. 8-6A are illustrated the symbols for enabled devices: simply a line with, typically, the letter G or E. If an inversion symbol is present, it means that this is an active low enable; if not, the line is an active high enable.

Clock lines are indicated with a small wedge-shaped figure, as shown in Fig. 8-6B. This symbol refers to edge-triggered operation, that is, the outputs change only on leading or trailing edges of the clock. Triggering on high to low transitions is given by an inversion symbol, and on low to high clock transitions without such a symbol.

EXPERIMENT 11, R/S LATCHES

Purpose

To observe the operation of gated and non-gated R/S latches.

Materials

1 - 74LS00	quad NAND
1 - 74LS279	quad R/S latch (optional)

Procedure

1. Construct the ungated R/S latch in Fig. 8-7A using two LS00 NAND devices. The game port connections are indicated as to both signal name and game socket number. Remember, you can construct the circuit with the computer on if you first connect ground to the bare chip, then the power Vcc, and only then the input and output lines (ANN and PB respectively).

2. Demonstrate the functions of this flip-flop. Note that when both inputs are inactive (high) the output state remains the same as the immediately preceding state, either set or reset. You can generate the

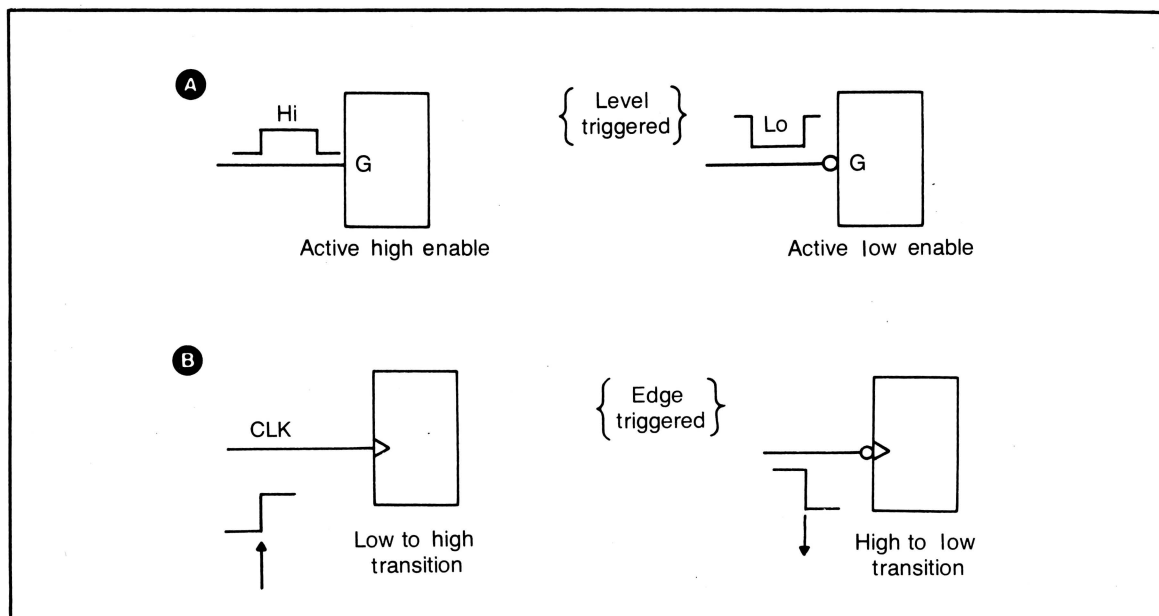


Fig. 8-6. Logic symbols for the (A) level-triggered latch and the (B) edge-triggered flip-flop.

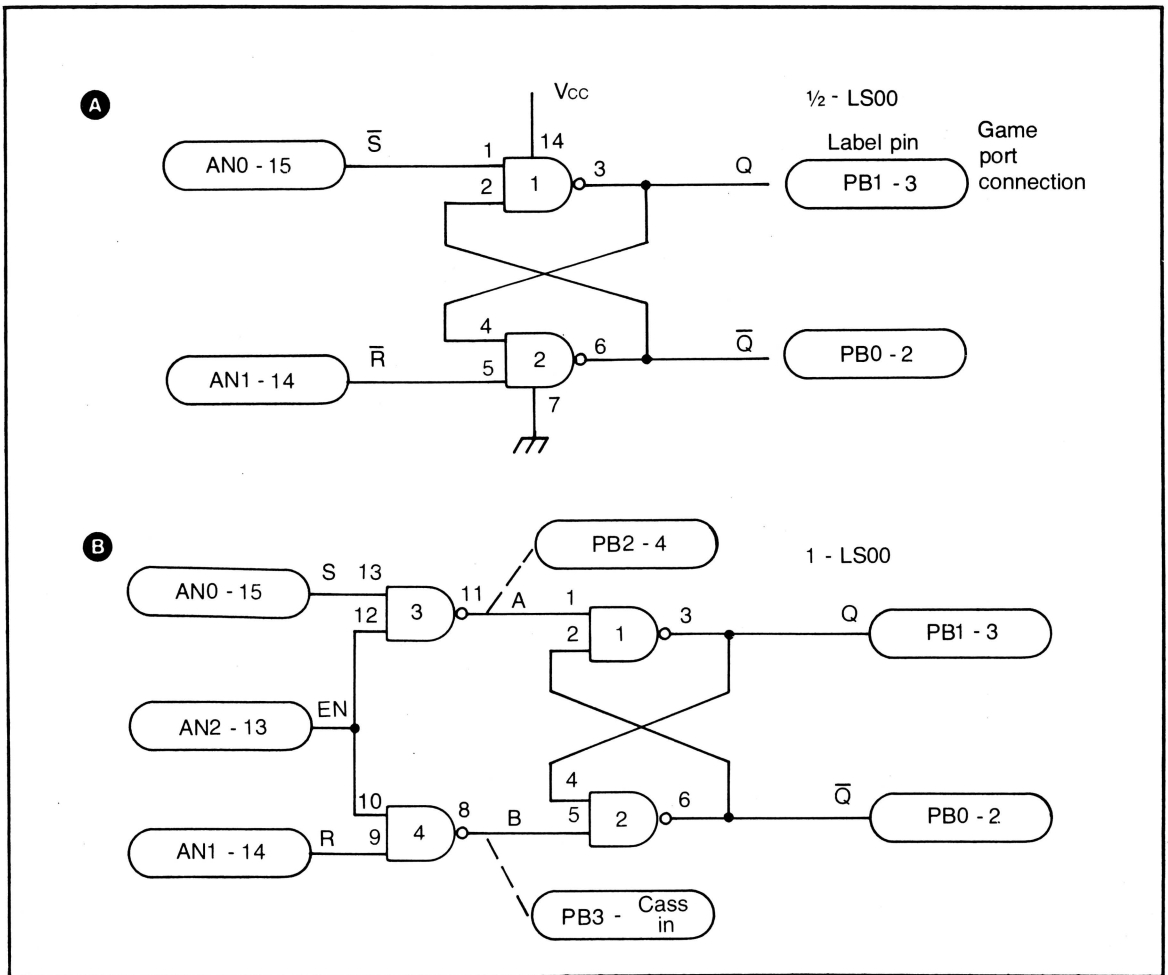


Fig. 8-7. Experiment 11 setups for the (A) ungated and (B) gated R/S latch using NAND gates.

state table for the ungated R/S latch with BDIS, as in Table 8-1. Unfortunately, you cannot show the ambiguity that results from illegal inputs (both low), because BDIS does not allow you to toggle two switches simultaneously.

3. Now hook up the circuit in Fig. 8-7B. You can leave the connections to the flip-flop section unchanged. Just add gates 3 and 4 to the configuration. You'll use all four PB lines: two for A and B and two for the final outputs Q and \bar{Q} . The fourth PB line is actually the cassette input line, as indicated. Note that an extra annunciator line is needed for the enable or clock line.

Table 8-1. Experiment 11 State Table for Ungated R/S Latch.

GP SIG:	AN1	AN0:	PB1	PB0
GP IN#:	14	15:	3	2
LABL1:	R'	S':	Q	Q'
LABL2:	5	1 :	3	6

0	0	0 :	1	1
1	0	1 :	0	1
2	1	0 :	1	0
3	1	1 :	1	0

Table 8-2. Experiment 11 State Table for Gated R/S Latch.

GPSIG:	AN2	AN1	AN0:	PB1	PB0
GPIN#:	13	14	15:	3	2
LABL1:	EN	R	S:	Q	Q'
LABL2:	12	9	13:	3	6

0	0	0	0 :	0	1
1	0	0	1 :	0	1
2	0	1	0 :	0	1
3	0	1	1 :	0	1
4	1	0	0 :	0	1
5	1	0	1 :	1	0
6	1	1	0 :	0	1
7	1	1	1 :	1	1

} Disabled

} Normal R/S operation

4. Confirm the action of this level-triggered latch or enabled flip-flop. Use the state table and timing diagram in Fig. 8-4 as guides. A state table comparable to Table 8-2 may be generated using BDIS.

5. Do the following. Disable the device with a low on EN(able). Place two highs on the R and S lines, and enable the device. You should see two highs on both Q and Q'. This is an illegal input. Now disable the device and place two lows on the R and S lines so that the flip-flop will be in a memory state when you enable it again. Now place a high on the EN line.

6. What state is the device in? It may be either set or reset. Repeat the sequence in step 5. In all probability, it will come on in the same state again, either set or reset.

Discussion

In the last two steps, the gated latch was used to show what happens when "illegal" (simultaneous) active signals are placed on the R and S lines. The procedure used was necessary because we could not realistically deactivate both simultaneously to find out what state the device would wind up in. (This is due to the nature of the Keyread routine in BDIS. However, you wouldn't be able to

do any better with mechanical switches either.) Of course, once back in the enabled memory state you will tend to wind up in the same state—set or reset—for a given circuit as you repeat the sequence in step 5. This is due to the slight electrical differences in the IC gates; one may conduct a tiny bit faster than another, or conduct a slightly stronger current. However, if you change the gate assignments around, or use another NAND package entirely, the outcome has even odds of being different.

The reason for emphasizing this point is not to caution you against designing a circuit which may produce illegal inputs into an R/S latch. Of course you would avoid this possibility. But you may require a fail-safe, high-reliability latching function in an application in which the chance of indeterminate inputs is totally unacceptable. In such cases, the simple R/S latch should probably be avoided. You would use one of the other flip-flops we will discuss which do not have the unpredictable, illegal input state.

Procedure (Continued)

7. Leave the circuit of Fig. 8-7B intact, as you may want to use it with slight modifications in the next section. With one modification, it can be made to operate as a D-type flip-flop.

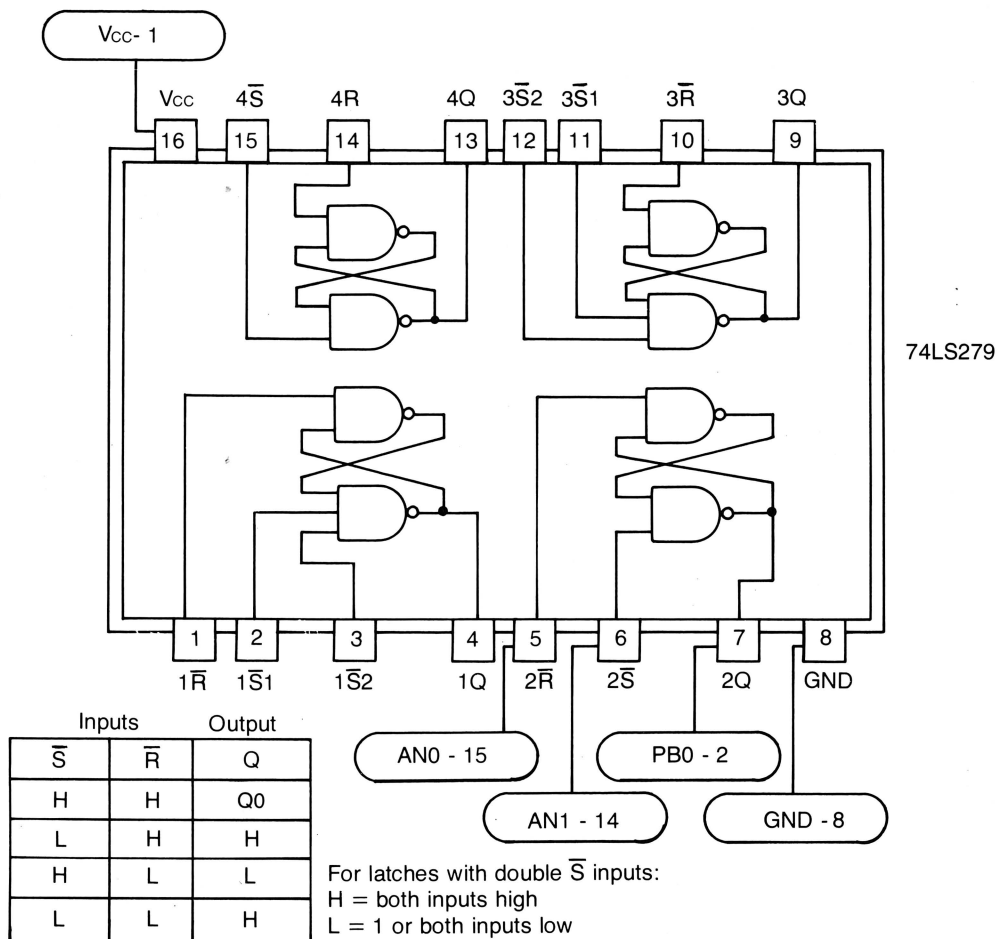


Fig. 8-8. Experiment 11 pin-out and state table for the LS279 equal R/S latch IC package.

8. Finally, there is an R/S latch in an integrated circuit form. The LS279 IC package illustrated in Fig. 8-8 contains four ungated R/S latches. Two of these latches have an extra set input. As an option, you may want to hook up this device and see how it functions. Use the game port assignments suggested in the figure. The state table for this hook-up will be as it appears in the figure. Note that H and L have supplanted 1 and 0, respectively, as you will see on the data manuals.

D AND T FLIP-FLOPS

This section introduces two more types of flip-flops that have several advantages over the R/S flip-flop. The D flip-flop is a very flexible device, and it warrants an inspection.

D-Type Flip-Flops

The D flip-flop or data flip-flop is just an extension of the enabled R/S device just discussed. In

fact, the only addition is an inverter, gate 5 in Fig. 8-9, which runs from the S input to the R input.

The D flip-flop has a single data input line, corresponding to the S input of the R/S latch, and an enable line. The purpose of the inverter is to assure that S and R are never both active high at the same time, thereby preventing the illegal state.

As you can see from the state table, the device is in a memory or retention state whenever the enable line is low. Otherwise, when enable is high, the outputs simply follow the data line transparently. The particular D-circuit shown is, then, a level-triggered device. This is indicated in the schematic by the letter G, for gated. The fact that there is no inversion symbol means that the line is active high. You'll also note that there is no illegal state.

If you wish, you can construct this circuit using four LS00 NAND gates and an inverter (one LS00 or one LS04).

Of course, D flip-flops can be constructed dif-

ferently, as edge-triggered devices. The logic symbol for such a device is illustrated in Fig. 8-10A. The clock line is denoted by the wedge-shaped symbol without an inversion circle. This means that the output transitions will occur on low to high transitions on the clock line. The clock signal must go low again before it can retrigger the device. It is important to emphasize that the output will contain the data as it occurs at the instant of low to high clock transition; data signals occurring after this leading edge will have no influence on the output, even though the clock voltage is still logic high.

The timing diagram of Fig. 8-10C is included to underscore these points. Note that with an edge-triggered device, there is no need for long clock pulses. In fact, they can be quite short, as pulse 2 in the figure.

T-Type Flip-Flops

T or toggle flip-flops, as the name suggests, alternate between high and low output states under

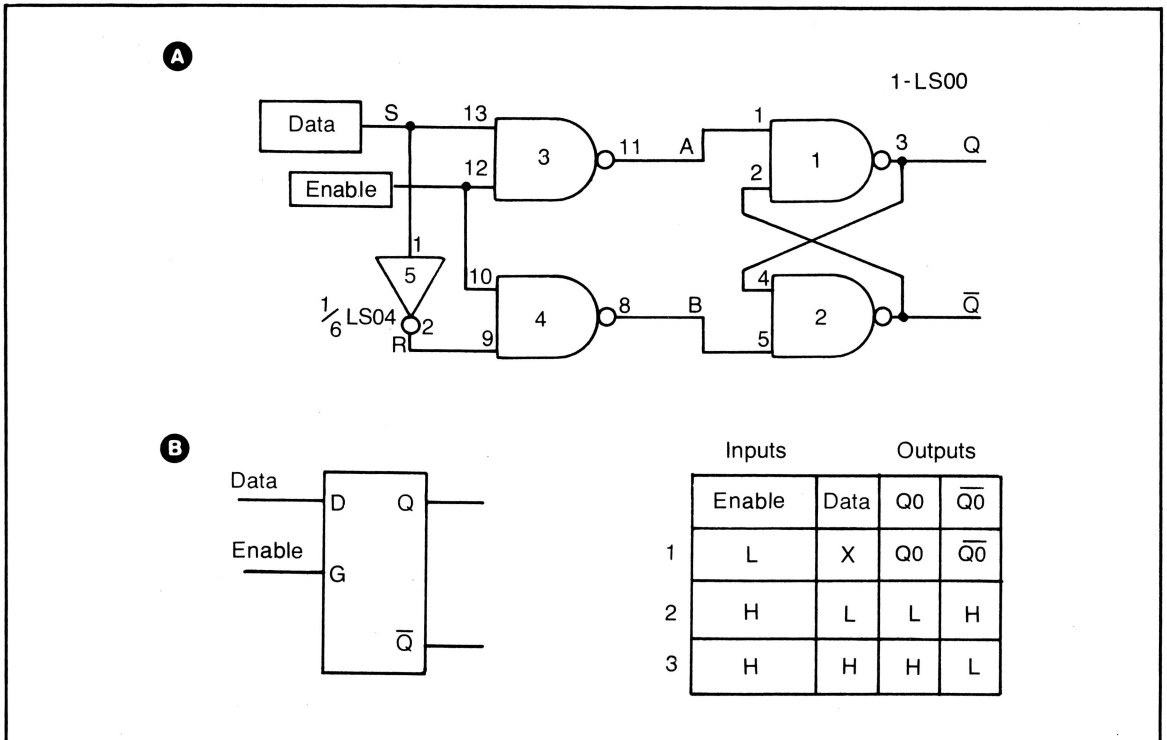


Fig. 8-9. The schematic, logic symbol, and state table for a level-triggered D-type flip-flop.

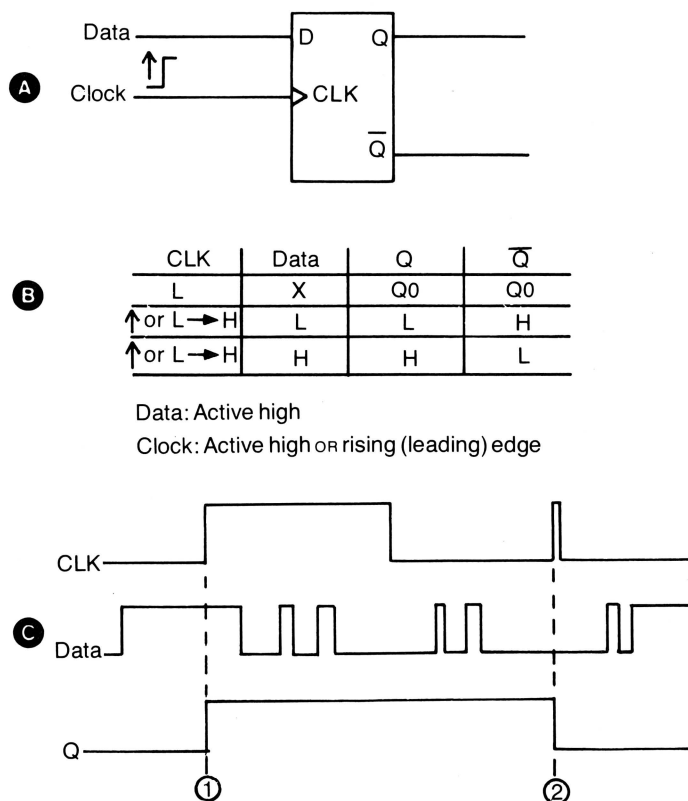


Fig. 8-10. Logic symbol, state table, and timing diagram for an edge-triggered D-type flip-flop.

the influence of a series of input pulses. The action is similar to that of certain mechanical pushbutton switches: push once and the switch is on, push a second time and it is off, push again and it is back on, etc.

The T flip-flop of Fig. 8-11A has an enable line (G). When high, any low to high transition on the toggle input line will complement the existing output state. On this particular device, we've included a clear input, so that we can start out from a known, low, output state if so desired. The timing diagram in Fig. 8-11B explains the operation of the device, namely, transition on every leading edge. For the waveforms shown, toggling takes place only during the active high of the G line. The device goes high on every other leading edge, so that for the four

toggle pulses shown (1 to 4), there are only two output pulses (A and B). The T flip-flop, then, acts as a simple divide-by-two device.

The state table of 8-11C is a concise method of expressing the above. The use of the symbol Q' in the second line means that the output will be complemented or inverted from its prior value on the up-going clock pulse.

Using the D as a T Flip-Flop

You won't find T flip-flops listed in the data manuals, because the toggling function can be realized by the D flip-flop, without external components. Typically, edge-triggered devices are used for this purpose.

Figure 8-12A illustrates how an edge-trig-

gered D flip-flop is set up to act in a toggle mode. Complimentary output Q' is fed back to the D input, and is the basis of the toggling action. The clock acts as the toggle line. Assuming the output begins at high/1, the first clock pulse will reset the Q output low, and the next one will set it high, as in 8-12B. At the end of four clock pulses, two leading edges will have been made at the output (pulses A

and B). This toggling action is identical to the T device just covered. The only difference between this and the device in Fig. 8-11 is that there is no enable line. If necessary, such a control line could be added by external gating.

Typical D-Type Devices

Two common SSI D-type flip-flops will serve

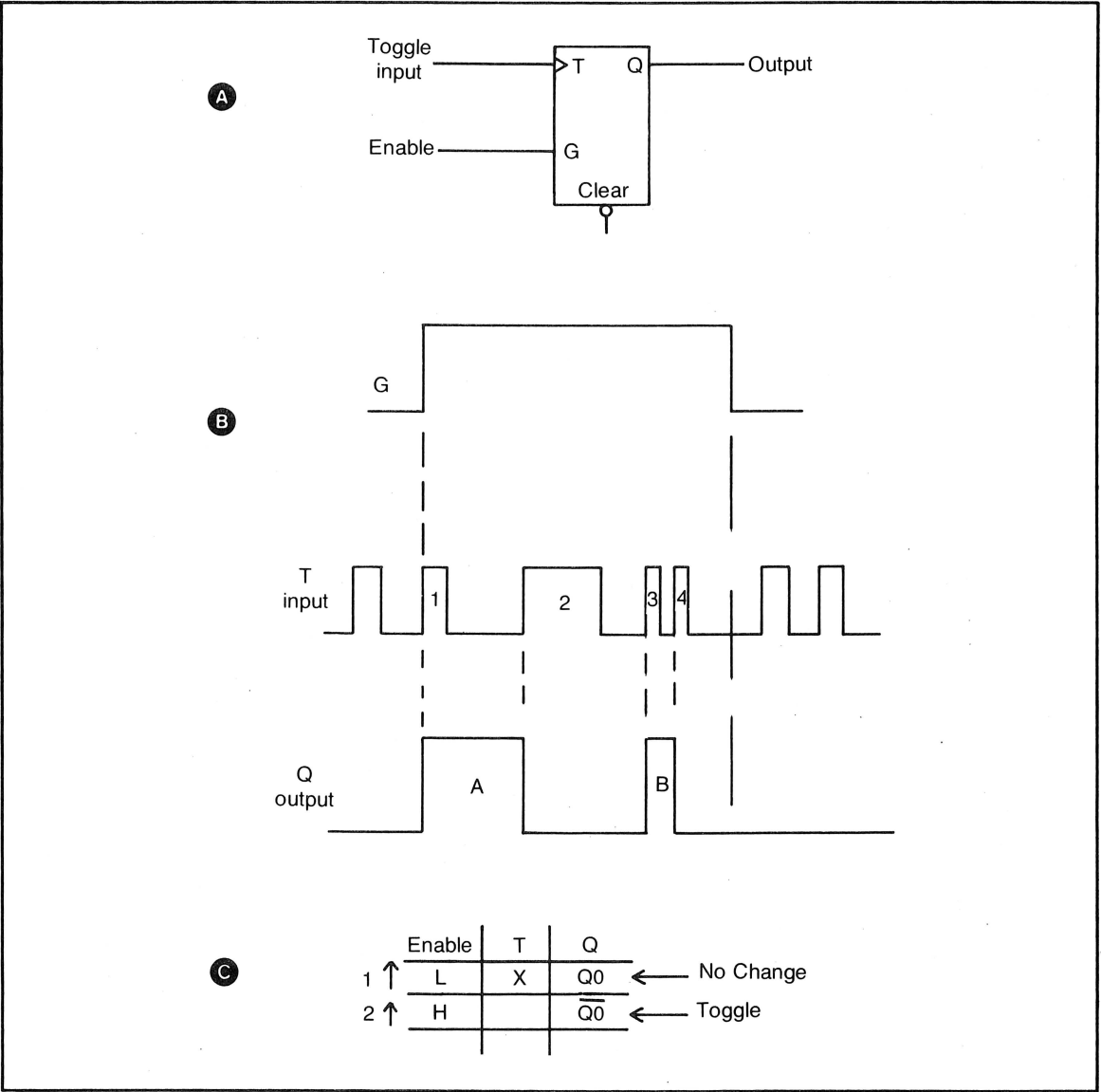


Fig. 8-11. Logic symbol, timing diagram, and state table for the toggle or T-type flip-flop.

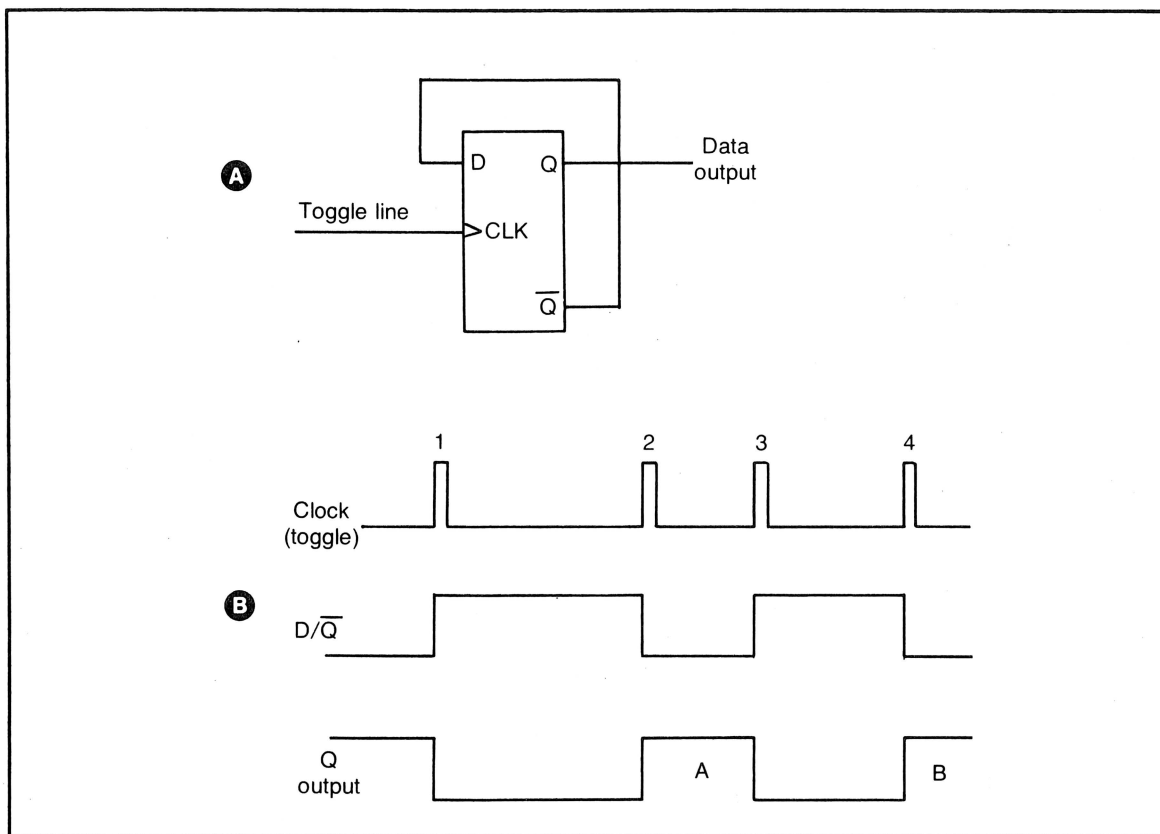


Fig. 8-12. Converting a D flip-flop into a T-type device.

as examples to underscore the above discussion. One is a transparent latch, and the other is an edge-triggered device.

The 74LS75 Quad Latch. The LS75 IC package contains four D-type latches which are level-triggered or transparent to data when enabled. The devices have dual-rail, or complementary, outputs. The devices are grouped in pairs, with each pair of latches enabled by a separate active high enable line (pins 4 and 13). The package pin-out is given in Fig. 8-13A. The state table in Fig. 8-13B is already familiar to you as is the logic symbol for this device in Fig. 8-13C.

Note in particular the placement of power and ground: pins 5 and 12, respectively. You probably also noticed that there are no clear lines to reset the outputs to zero. One reason for this is that, as a latch, the device is intended to temporarily store

data, not to manipulate it. In addition, the 16-pin package is of a particularly convenient size, and additional lines would increase package size with little real gain in flexibility.

The 74LS74 Dual-D Edge-Triggered Device. If you require a full-featured D-type flip-flop, then the LS74 may be appropriate. Its pin-out is shown in Fig. 8-14A. This is an edge-triggered device, the output changing state on the low to high clock transition. Like the LS75 it is a dual-rail flip-flop.

In addition, the LS74 is provided with both preset and clear inputs, allowing the Q output to be set high or low, respectively. These two lines are *asynchronous*, because they act independently of the clock. They act much like the set and reset lines on the R/S latch and provide a bit more flexibility.

The *synchronous* lines are the data inputs to

the two flip-flops on this package. They influence the output state only on the up-going edge of the clock signal.

EXPERIMENT 12, THE LS74 FLIP-FLOP

Purpose

To present the operation of a typical edge-triggered D-type device and an application.

Materials

1 - 74LS74 dual-D flip-flop

Procedure

1. We'll take a look at the LS74 edge-triggered device first. Using Fig. 8-15 as a guide, wire up one-half of an LS74 on the solderless breadboard. Connect first ground and then power pins to game

connector pins 8 and 1, respectively. Then, annunciators 0, 1, 2, and 3 are connected to preset, clear, data, and clock pins. (The ellipses contain the game port symbol name and pin number on the game connector, as in the previous experiment.) Connect PB0 and PB1 to the complementary inputs, as shown.

2. You can set up LABL1 on the BDIS display to correspond to the LS74 signal names (DAT, CK, Q, Q', etc.). Use LABL2 for the device pin numbers.

3. Verify that the preset and clear lines operate just like the R' and S' lines on an R/S latch. Table 8-3A reflects the R/S function of these lines. Table 8-3B shows that both lines override both clocks and data inputs, in particular, setting the outputs high whatever the state of clock and data.

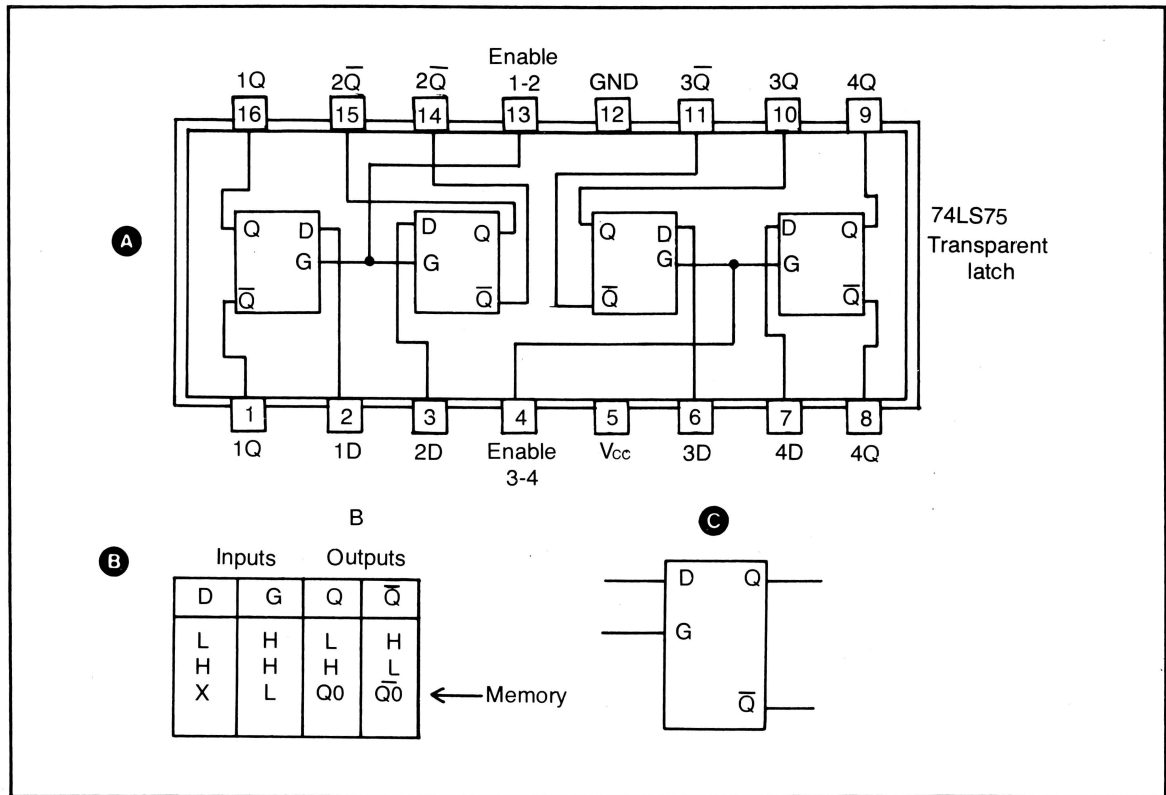


Fig. 8-13. A popular level-triggered or transparent D-latch, the LS75.

Table 8-3. Experiment 12 State Table That May be Generated for the LS74 Edge-triggered Latch.

A

GPSIG:	AN3	AN2	AN1	AN0:	PB1	PB0
GPIN#:	12	13	14	15:	3	2
LABL1:	PR'	CL'	CLK	DAT:	Q	Q'
LABL2:	4	1	3	2 :	5	6

0	0	0	0	0 :	1	1
1	0	1	0	0 :	1	0
2	1	0	0	0 :	0	1
3	1	1	0	0 :	0	1

B

GPSIG:	AN3	AN2	AN1	AN0:	PB1	PB0
GPIN#:	12	13	14	15:	3	2
LABL1:	PR'	CL'	CLK	DAT:	Q	Q'
LABL2:	4	1	3	2 :	5	6

0	0	0	↑ 1	0 :	1	1
1	0	0	↑ 1	1 :	1	1

C

GPSIG:	AN1	AN0:	PB1	PB0
GPIN#:	14	15:	3	2
LABL1:	CLK	DAT:	Q	Q'
LABL2:	3	2 :	5	6

0	0	0/1:	0	1
1	↑ 1	1 :	1	0
2	1	0/1:	1	0
3	0	1 :	1	0
4	↑ 1	0 :	0	1
5	1	0/1:	0	1
6	0	0/1:	0	1

4. Now tie present and clear to +5 volts for the rest of the experiment. This can be done by simply attaching these lines directly to VCC. Then verify that the logic level on the data line (AN2 in the figure) is latched only on the rising edge of the clock

signals. Outputs Q and Q' reflect the data in non-inverted and inverted form, respectively. Table 8-3C indicates the type of result you should achieve. Upward arrows in the clock columns signify proper triggering; the 0/1 in the data column signifies that

toggling of data has no affect when the clock is unchanging.

5. Now connect the LS74 as shown in Fig. 8-16. Both devices on this dual-D package are used. Make sure both PR' and CLR' are high before proceeding.

6. To generate a state table, set up the headings as suggested in Table 8-4. Toggle the clock line (AN2) until both outputs (Q1' and Q2') are low/0, leaving

CLK low, as in line 0 in the table. Press RTN and toggle CLK high, line 1, and then press RTN and toggle CLK low, as in line 2. Press RTN again, and toggle CLK high, and then RTN, CLK low, lines 3 and 4. Continue until you've generated the 17-line table shown. What function is this circuit performing?

Discussion

In the first part of the experiment you demonstrated the basic operation of the LS74 edge-

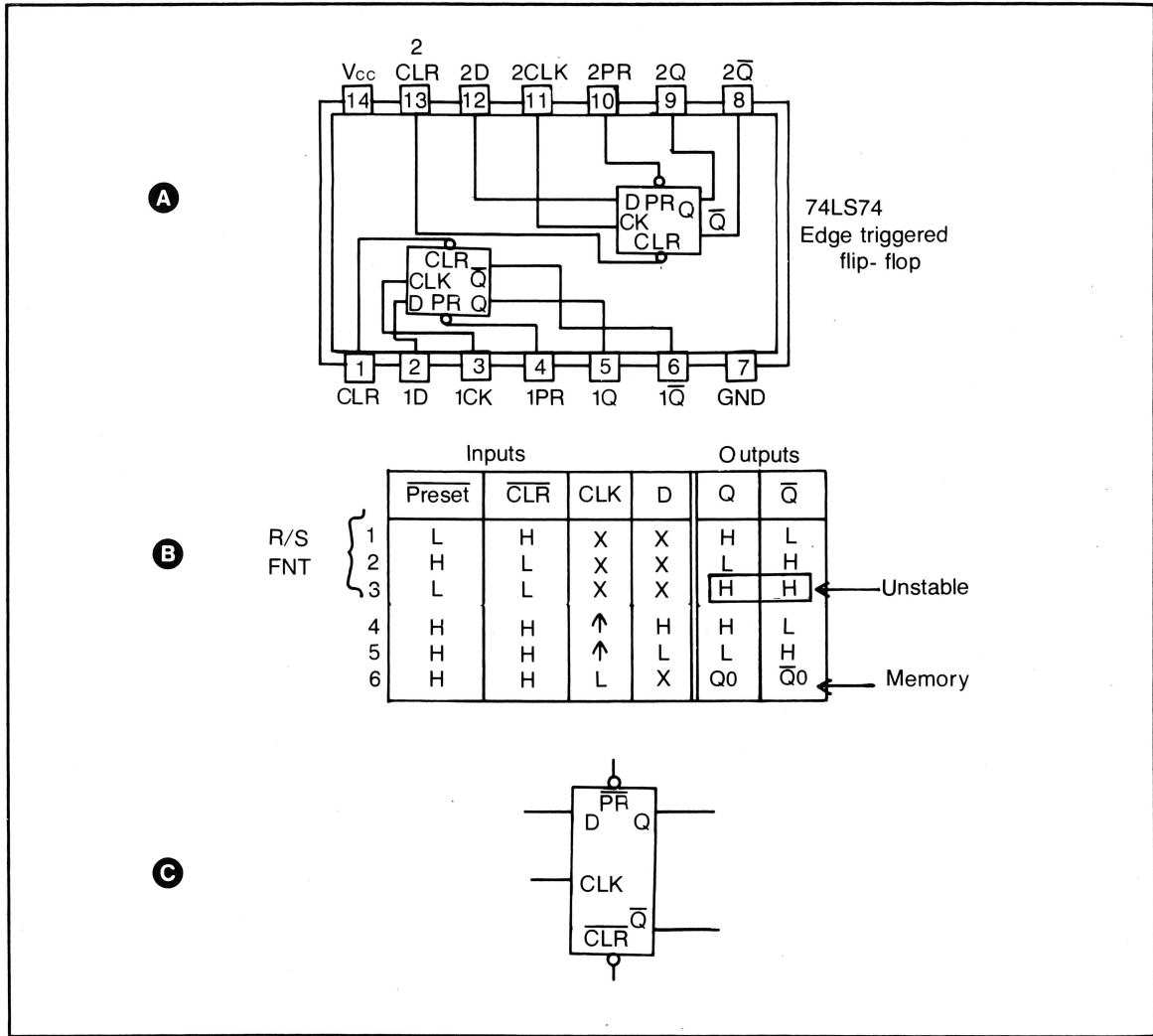


Fig. 8-14. A popular edge-triggered flip-flop, the dual-D LS74.

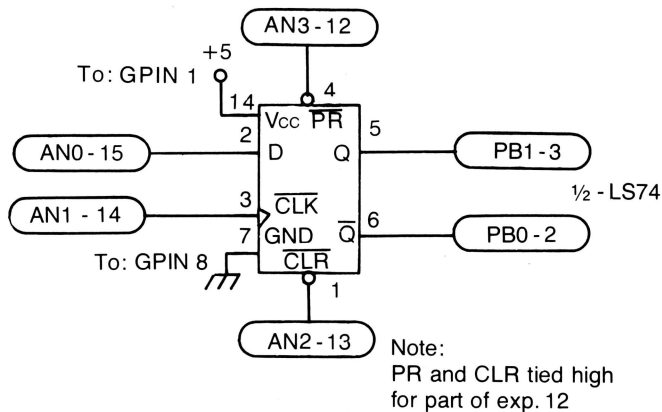


Fig. 8-15. Experiment 12 setup of LS74, basic operation.

triggered device. In the second part, you build a circuit which performed a counting function. Specifically, this is a two-bit counter, which counted from binary 00 to 11, or decimal 0 to 3. (The reason the clear and preset lines were inactivated by tying them high was to facilitate the generation of the state table using BDIS).

What other function does the circuit in Fig. 8-16 perform? Well, it can be alternatively thought of as a frequency divider. That is, if the clock was driven by a source of regular square waves of frequency f_c , then $Q1'$ and $Q2'$ would be alternating between high and low at rates of $\frac{1}{2}$ and $\frac{1}{4}$ of f_c ,

respectively. This is indicated to the right of Table 8-4, by the corresponding timing diagram.

Counting and frequency division, along with timing operations, are important functions of sequential devices. We'll say more about these applications at both the SSI and MSI levels of complexity.

EXPERIMENT 13, THE LS75 QUAD D-LATCH

Purpose

To illustrate the operation of a transparent or level-triggered flip-flop, along with an application.

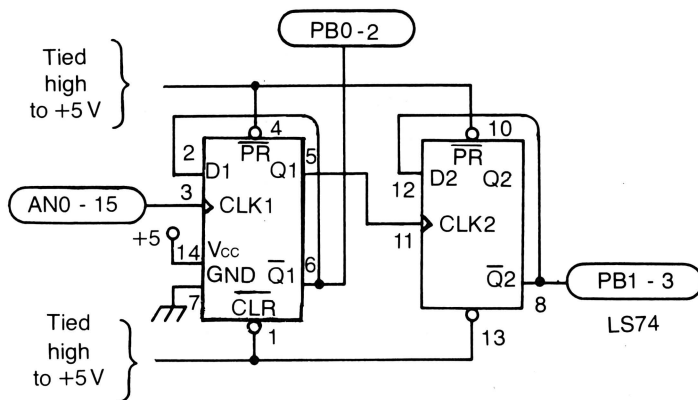
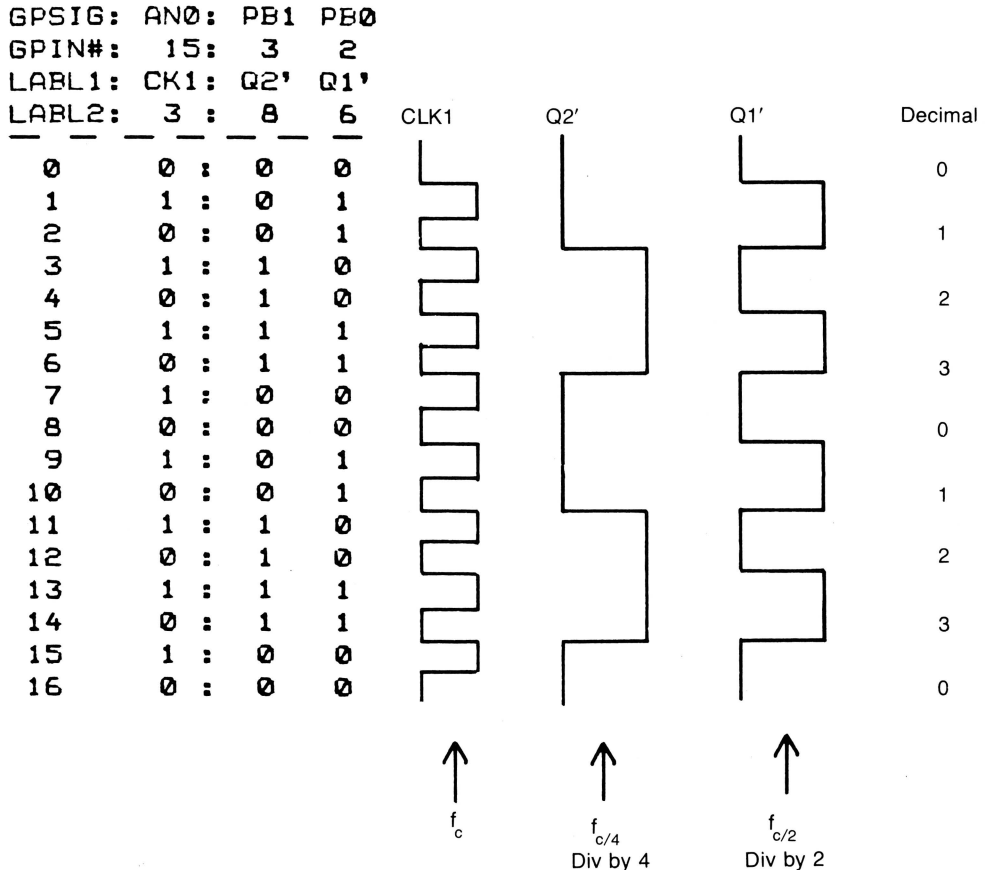


Fig. 8-16. Experiment 12. Use of LS74 as a divide-by-four, or two-bit counter.

Table 8-4. Experiment 12 Combined State Table and Timing Diagram for the Two-bit Counter of Fig. 8-16.



Materials

- 1 - 74LS75 quad D-latch
- 1 - 74LS74 dual-D flip-flop
- 1 - LED
- 1 - 330 ohm resistor

Procedure

1. Wire up the circuit in Fig. 8-17. Again, with the computer on, connect GND +5 V VCC, and ANN/PB lines to and from the chip, in that order.
2. With this level-triggered flip-flop, a low on G will disable the device—changes on the data (D1)

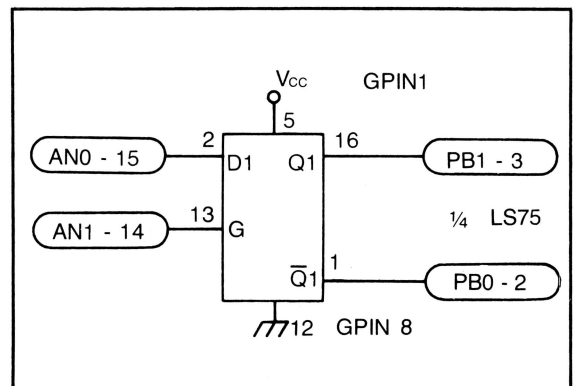


Fig. 8-17. Experiment 13. Basic setup of the LS75.

Table 8-5. Experiment 13 State Table for the LS75 D Latch.

GPISIG:	AN1	AN0:	PB1	PB0
GPIN#:	14	15:	3	2
LABL1:	G	D1:	Q1	Q1'
LABL2:	13	2 :	16	1

0	0	0 :	0	1
1	0	1 :	0	1
2	1	0 :	0	1
3	1	1 :	1	0

line will have no influence on the output. With G high, the device is enabled, and is transparent to data—Q1 follows D1, with Q1' the complement or inverse of Q1. You may want to generate the state table for the device, with appropriate headings, as suggested in Table 8-5.

3. Examine the circuit in Fig. 8-18. What function does it perform? Hook up this circuit as shown. You may want to wire up the LS74 first, and verify that it performs correctly as a toggle flip-flop. (It should work with the PR' and CLR' lines floating high. If not, tie these lines directly to VCC.) Using the LS75

package, make the connections as shown in the figure.

4. Toggle the UTILSTB (letter S on the keyboard) so that the LED is off. With the LED unlit, you know that the device is disabled. (Why?) Now toggle the data lines, D1 through D4, high and low. As you'd expect, data does not pass through to the corresponding outputs, Q1 and Q4.

5. Toggle the UTILSTB so that the LED is lit. The device is now enabled. Now if you toggle any of the data lines, the changes are reflected directly at the

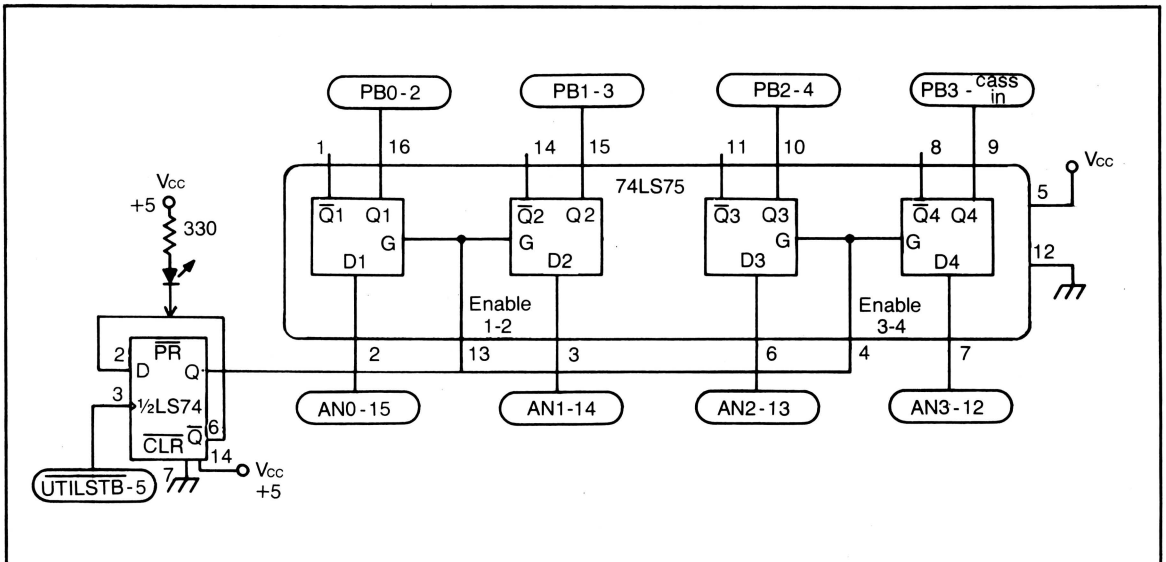


Fig. 8-18. Experiment 13. Use of LS75 as a four-bit register.

Table 8-6. Experiment 13 Tabular Record of the LS75 Package Used as a Four-bit Latch or Register.

		GPSIG:	AN3	AN2	AN1	AN0:	PB3	PB2	PB1	PB0
		GPIN#:	12	13	14	15:	CS	4	3	2
		LABL1:	D4	D3	D2	D1:	Q4	Q3	Q2	Q1
		LABL2:	7	6	3	2 :	9	10	15	16
		<hr/>								
Disabled	{	0	0	0	0	1 :	0	0	0	0
		1	0	0	1	0 :	0	0	0	0
		2	0	1	0	0 :	0	0	0	0
		3	1	0	0	0 :	0	0	0	0
Enabled	{	4	0	0	0	1 :	0	0	0	1
		5	0	0	1	0 :	0	0	1	0
		6	0	1	0	0 :	0	1	0	0
		7	1	0	0	0 :	1	0	0	0

corresponding outputs. This is represented by the state table in Table 8-6.

Discussion

In the first circuit you demonstrated the basic operation of the LS75 latch.

In the second part of the experiment, for the first time, you had to use all the available digital input and output lines available on the game port: five output lines consisting of four annunciators and the utility strobe, plus the four pushbutton lines. You needed a high/low signal as the fifth annunciator in order to conveniently perform the experiment. Therefore in order to use the $\frac{1}{2} \mu\text{sec}$ pulse of the UTILSTB, you had to make it turn some device on/high and off/low on alternate keypresses. The solution was to use the LS74 as a T-type flip-flop.

The LED was then attached to Q'. The reason for this is simple. Whenever Q' was low and sinking current through the LED, the LED would light up. Under these conditions Q would be high, enabling the LS75.

In conclusion, the second part of this experiment illustrated the basic application of the LS75 as a transparent latch which holds data in parallel form, in the case as a 4-bit *nibble*. Two such devices in parallel would latch or hold eight bits of data, a full data word or byte. Similar but more complex

MSI chips exist which also perform latching functions; some are dedicated to bus-oriented data transfer. We'll talk about these in a later chapter.

THE J-K FLIP-FLOP

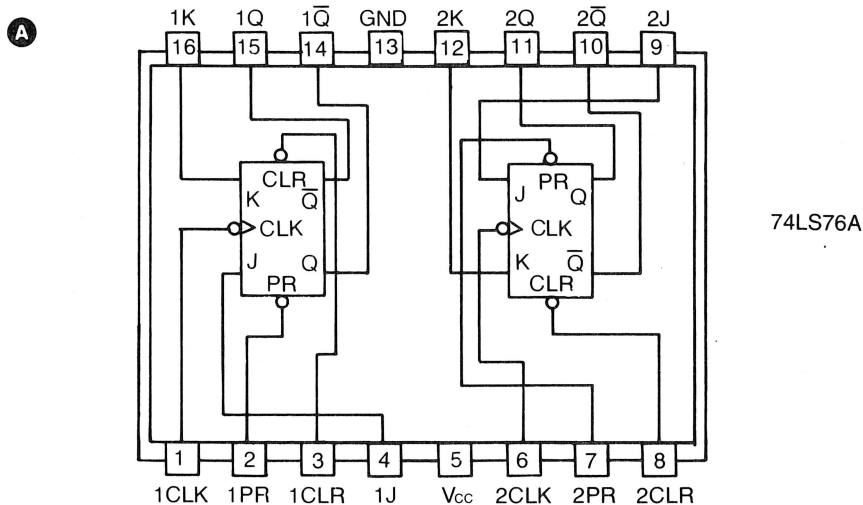
The last flip-flop we'll examine is the J-K flip-flop. This is the most versatile of the SSI bistable devices because it performs all the functions of the flip-flops just covered and has a few added features of its own. We will look at a specific device, the 74LS76A. This representative J-K flip-flop can be best understood by comparison to the other devices mentioned. It performs the following functions:

☐ It can as an R/S latch. That is, it possesses both preset and clear lines which act like the R' and S' lines on the simple R/S flip-flop.

☐ It acts as an edge-triggered D-type flip-flop, much like the LS74. The only difference is that output transitions occur on the trailing edge or high-to-low transition of the clock.

☐ It can be disabled, so that it retains the current output state regardless of the state of input J and K lines or of the clock line. This is unlike the LS74, which does not have a separate enable.

☐ Finally, the J-K device can act as a T-type flip-flop. It can be set in a toggle mode without any external wiring. This is another advantage over the LS74.



Inputs						Outputs	
	Preset	Clear	Clock	J	K	Q	\overline{Q}
R/S function	1	L	H	X	X	X	L
	2	H	L	X	X	X	H
	3	L	L	X	X	X	H
							← Unstable
4	H	H	↓	L	L	Q0	$\overline{Q}0$
5	H	H	↓	L	L	H	L
6	H	H	↓	L	H	\overline{L}	H
7	H	H	↓	H	H	$\overline{Q}0$	Q0
8	H	H	H or L	X	X	Q0	$\overline{Q}0$
							← Memory
							← Toggle

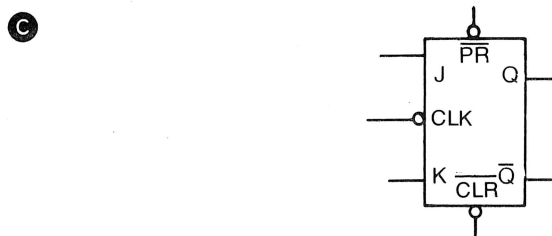


Fig. 8-19. Pin-out, state table, and logic symbol of the LS76A J-K flip-flop.

The pin-out of the LS76 is shown in Fig. 8-19A. It is a 16-pin device with two J-K flip-flops per package. Power and ground are on pins 5 and 13, respectively. Each flip-flop (Fig. 8-13C) has a

reset and clear line, J and K inputs for data plus a clock line, and complementary (dual-rail) outputs.

The state table in Fig. 8-19B explains device operation. Lines 1 to 3 show the R/S function. In

these three states, the clear and preset lines override the clock and J and K lines. Lows on both of the asynchronous lines are, of course, an undesirable state.

In lines 4 to 8, both clear and preset are high, that is, inactive. In this state the mode of device operation is determined by the logic levels on the J and K inputs.

□ Line 4 memory mode—If both J and K inputs are low, then the flip-flop is disabled. It is in a memory state. The clock has no influence.

□ Lines 5 and 6 data mode—If J is high/1 and K low/0, the output Q will be high. Conversely, when J is low and K is high, Q is low. In this way data is transferred to the dual-rail output. Output transition occurs on the down-going edge of the clock, as indicated by the arrow.

□ Line 7 toggle mode—with both J and K high/1, the device acts as a toggle flip-flop, complementing its output state on each falling edge of the clock pulse.

These then are the main modes of edge-triggered J-K operation. Line 8 in the state table is included merely to emphasize that the device will not change state when the clock is stable (either high or low), but only on a down-going edge.

This reason for trailing edge transition needs explained. First, you should note that the J-K device consists of two flip-flop sections which operate on the *master-slave* principle. It consists of a control or master flip-flop on the input end, and a slave flip-flop section on the output end, as seen in Fig. 8-20. The internal details of the circuitry involved are not important for our purposes. However, the

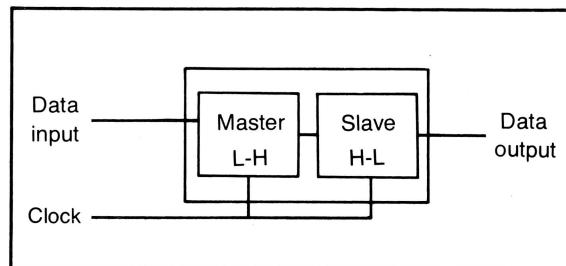


Fig. 8-20. The master-slave principle behind J-K devices.

two-stage operation of the flip-flop is significant as regards when output transitions occur.

Assume that the initial output state of the flip-flop is low, that is, $Q = 0$. This is illustrated in the timing diagram of Fig. 8-21. When operating in the data mode the J-K inputs will influence the output state only during clock transitions, provided that J and K are unequal, denoted by J and K' in the figure. As shown in this timing diagram, data pulses occurring during the time when the clock line is high or low have no influence on the output. Now, at data pulse 1, J is high and K is low. When the clock goes high at this time, the master flip-flop state undergoes transition from low to high. The Q output from the slave section remains low. The clock pulse must return to low before the slave flip-flop section will undergo transition to the high state. Again, pulses 2 and 3 between these two edges have no influence on the output.

In short, the master/input section is set by the data on the leading edge and the slave/output section is set on the trailing edge of the clock pulse. This is indicated by the LE and TE labels.

EXPERIMENT 14, THE J-K FLIP-FLOP

Purpose

To demonstrate the basic operation of a typical J-K flip-flop, the 74LS76A, and to show how it is configured for one of its principle applications, namely, as a counter.

Materials

1 - 74LS76A	quad J-K flip-flop
1 - 74LS74	dual D flip-flop
1 - 74LS00	quad NAND
1 - LED	
1 - 330 ohm resistor	

Procedure

1. The circuit depicted in Fig. 8-22 will allow you to explore the main features of the LS76A. J-K flip-flop. Again, you'll use $\frac{1}{2}$ of an LS74 to allow you to alternate the clock between high and low. This is not strictly necessary, as the experiment can just as easily be performed using the utility strobe pulse directly as it comes off the game port connector.

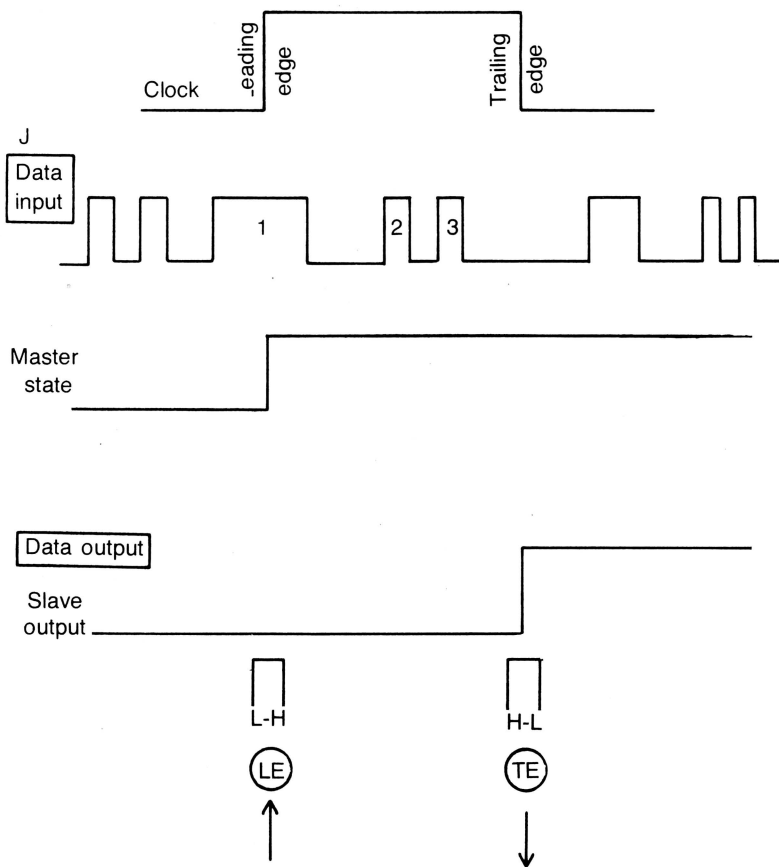


Fig. 8-21. The timing diagram for the falling-edge transition of the master-slave flip-flop.

However, the LS74 circuit gives you extra control over the duration of the strobe pulse, as well as a visual indicator of the clock level.

2. Observe how the preset and clear inputs override the J and K inputs. Lines 0 to 2 in the state table of Table 8-6A indicate the conditions of override. In line 3, these asynchronous inputs are inactive, and normal J-K operation can take place. After verifying this, tie the clear and preset inputs to high (VCC, +5 V) before proceeding to step 3. This will allow you to generate the J-K state table.

3. After tying PR' and CLR' high, verify the four modes of J-K operation from the discussion in the

section on the J-K flip-flop. In retention or memory mode, toggling the clock will have no affect. When J is not equal to K, the Q output will be either low/0 or high/1, depending on the J-K values. Output transition occurs on the down-going edge. With J and K = 1, the device will toggle between high and low output on alternate clock pulses. You can record this sequence of events using BDIS. A typical result might look like that in Table 8-6B.

4. Analyze and then build the circuit in Fig. 8-23. You would suspect that it is a counter of some sort. The best way to prove this is to use a timing diagram, beginning with the assumed status of all outputs equal to zero. Such a diagram is given in Fig.

Table 8-7. Experiment 14. (A) Overriding Preset and Clear Functions of the J-K Flip-Flop (B) The Four Modes of J-K Operation in Tabular Form with Timing Diagram.

(A)

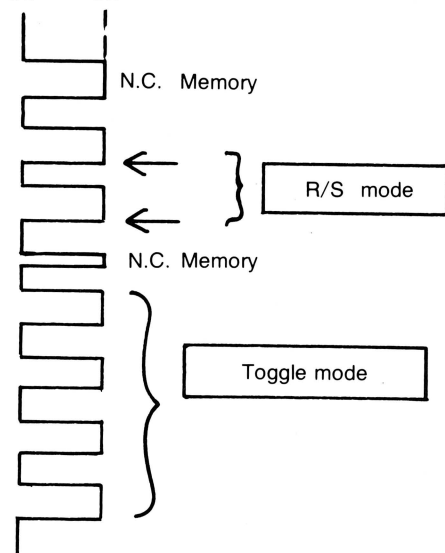
GP SIG:	AN3	AN2	AN1	AN0:	PB1	PB0
GP IN#:	12	13	14	15:	3	2
LABL1:	PR'	CL'	J	K :	Q	Q'
LABL2:	2	3	16	4 :	15	14

0	0	0	0/1	Q/1:	1	1
1	0	1	0/1	Q/1:	1	0
2	1	0	0/1	Q/1:	0	1
3	1	1	0	0 :	0	1

(B)

GP SIG:	AN1	AN0:	PB1	PB0	UTILSTB
GP IN#:	14	15:	3	2	light
LABL1:	J	K :	Q	Q'	Off On
LABL2:	4	16:	15	14	

0	0	0 :	0	1	
1	0	0 :	0	1	
2	1	0 :	0	1	
3	1	0 :	1	0	
4	0	1 :	1	0	
5	0	1 :	0	1	
6	0	0 :	0	1	
7	1	1 :	0	1	
8	1	1 :	1	0	
9	1	1 :	1	0	
10	1	1 :	0	1	
11	1	1 :	0	1	
12	1	1 :	1	0	
13	1	1 :	1	0	
14	1	1 :	0	1	



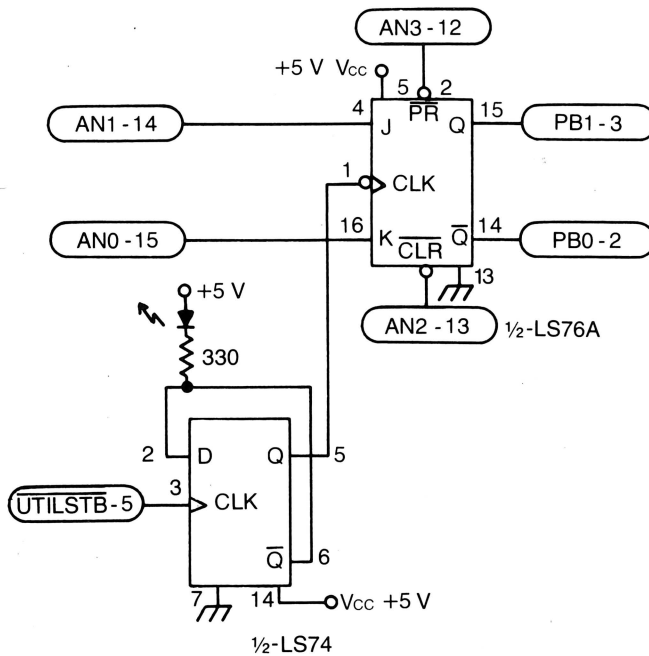


Fig. 8-22. Experiment 14 setup for basic LS76A operation.

8-24. To operate this circuit, J and K should be high/1 and clear should be high as well. Pulsing the clock line (AN0) will toggle the first flip-flop. Operate the circuit on BDIS. How would you describe

its function? Do not dismantle this circuit.

5. Now analyze and build the circuit in Fig. 8-25 by making the small addition of a NAND gate. At some

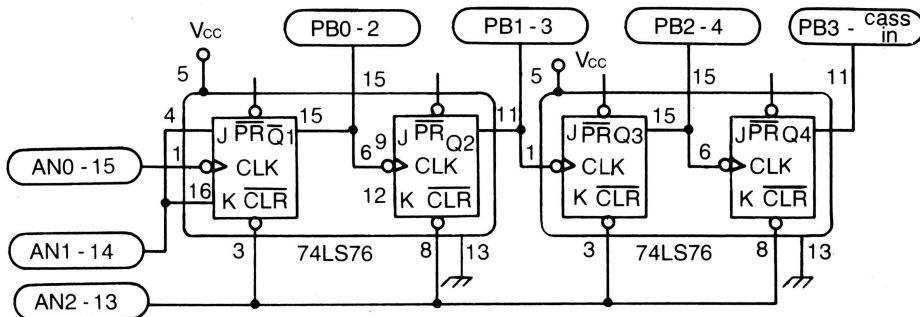


Fig. 8-23. Experiment 14. Use of two LS76A packages as a four-bit binary counter.

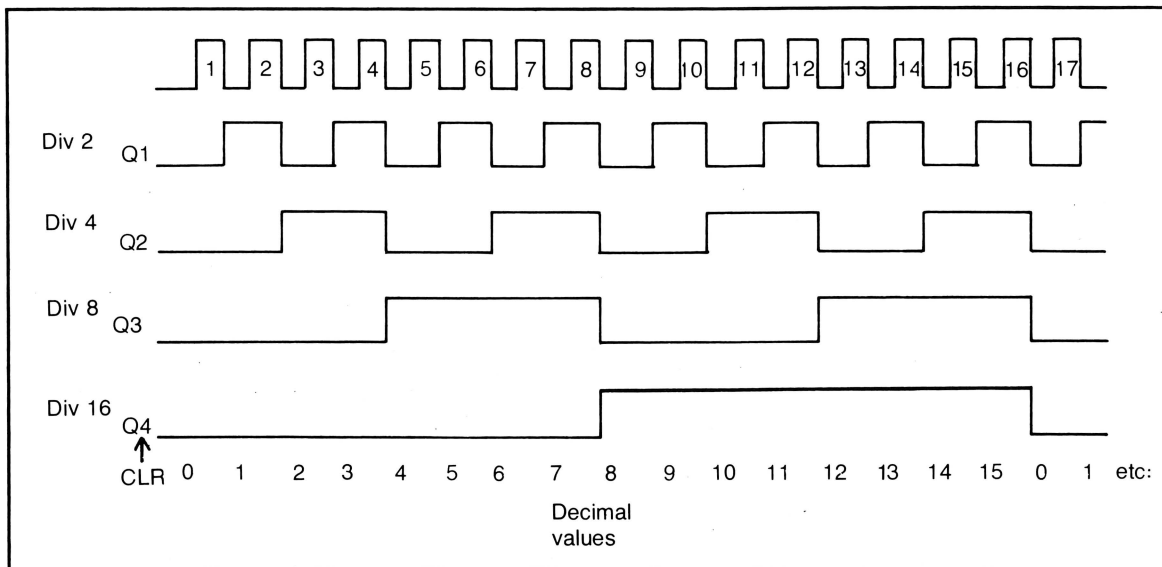


Fig. 8-24. Experiment 14. Timing diagram for the four-bit counter.

point in the count sequence, the clear line will go low. When? What type of counting operation does this circuit perform? Again, use a timing diagram analysis, as suggested in Fig. 8-26.

Discussion

Regarding the functions performed by the above circuits, it should be clear to you that the counters you constructed were a four-bit binary

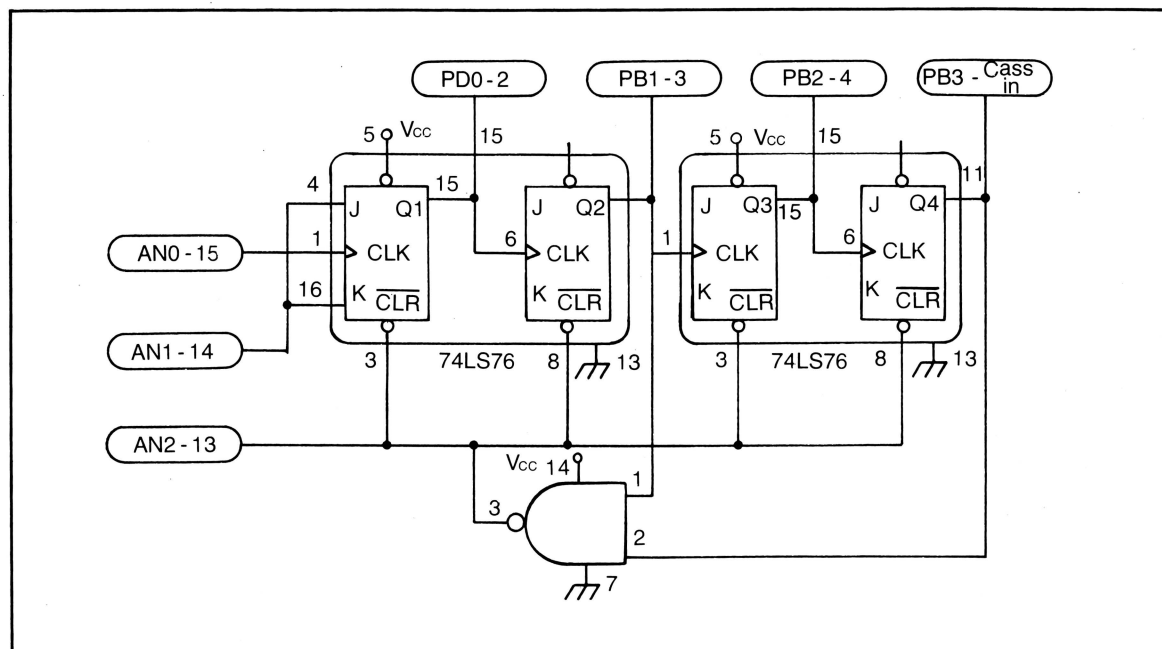


Fig. 8-25. Experiment 14. Use of two LS76A packages as a one decimal digit BCD counter.

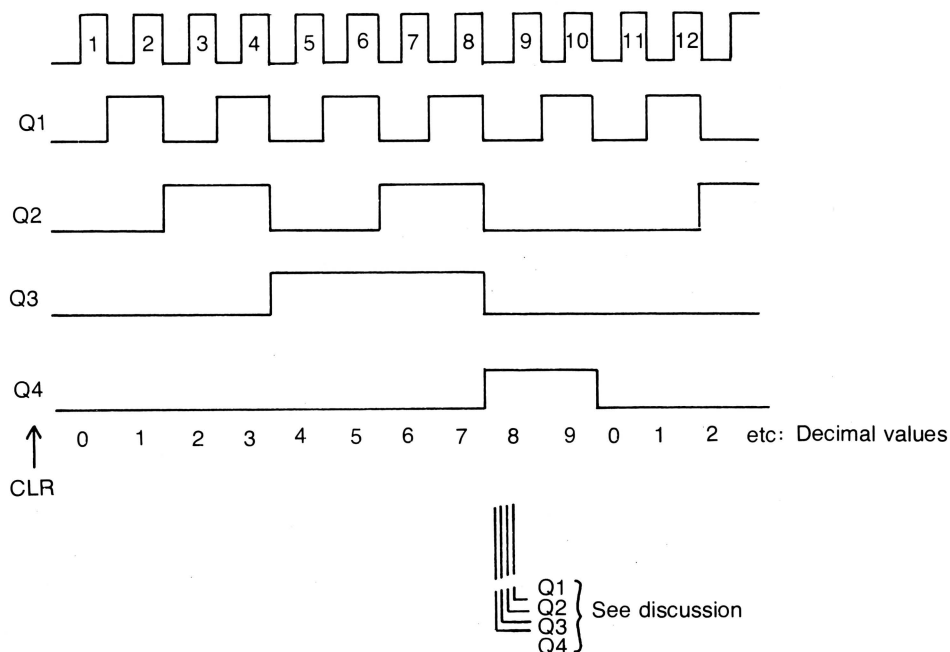


Fig. 8-26. Experiment 14. Timing diagram for the BCD counter. The transitions in this and the previous ripple counter are not simultaneous, as indicated at the trailing edge of the 8th clock pulse.

counter in step 5 and a binary coded decimal or BCD counter in step 6. The binary counter could count from 0000 to 1111 (decimal 0 to 15). The BCD counter could count from 0 to 9, that is one decimal digit.

The BCD code is nothing more than the 0000 to 1001 sequence, with numbers between 1010 and 1111 disallowed as illegal. As you saw, the BCD counter never reached 1010 (decimal 10) but simply recycled back to zero. BCD counters may be used as part of the display hardware to drive seven-segment LED and LCD (liquid crystal) readouts in test equipment, clocks and watches, etc.

There are a few additional points to be brought out.

First, the counters you constructed from individual J-K flip-flops in the last two steps are known as *ripple counters*. If you analyzed the operation of these counters, you could see that the clock line of flip-flops 2, 3, and 4 were connected to the Q output

of the preceding flip-flop. That is, these flip-flops did not change state until the preceding flip-flop changed state. Because of the expected propagation delay between input and output in each device, this means that on each count pulse there is a ripple affect; the flip-flops in the counting chain do not change simultaneously on the down-going clock pulse.

This is indicated on the falling edge of pulse 8 in Fig. 8-26, a magnified view of the Q1 through Q4 transitions at that instant. Ripple counters are simple, but they have limitations due to the nonsimultaneity of the output transitions.

The second major point is that building even a simple 4-bit counter from SSI sequential components—one with only clock, combined J-K, and clear inputs—is a bit of a pain. You have to make a number of interconnections between the individual flip-flops, and you wind up with a rather limited set of functions. If you wanted to overcome the ripple

problem, the external circuitry would become more complex.

The cure to both problems is found in the realm of medium scale integration sequential devices. MSI synchronous counters and other powerful devices are covered in the next chapter.

MONOSTABLE DEVICES OR ONE SHOTS

The idea behind monostable devices, or one shots, is very simple. When triggered by a short pulse, they output a square wave with the duration determined by external components. The trigger pulse may be either the rising or falling edge of a square wave. The external components are nothing more than an RC combination. The duration of the output pulse is known as the pulse width, or PW in Fig. 8-27.

Building a one shot from SSI components is possible. For the purposes of discussion, we'll do this on paper. Figure 8-28A shows a one shot consisting of a J-K flip-flop and five inverters. The timing diagram of Fig. 8-28B explains the operation. Assume the stable state is one in which J is high and K is low. Along comes a clock pulse and after one flip-flop propagation delay, the output goes high. This output is in turn propagated through an odd number of inverters, in this case five, with the net result that the clear line goes low after five inverter propagation delays. After one more flip-flop delay, the Q output goes low following the clear. During the entire time, the output has re-

mained high, giving a stretched out version of the initial trigger pulse. Computation of the total duration of the pulse is given in Fig. 8-28C. Note that $J=1$ and $K=0$ is the stable state of this device.

Another similar circuit, with an even number of inverter delays, is shown in Fig. 8-29. Operation is similar to that of the previous circuit.

Problems

There are several problems with, or limitations to, the circuits illustrated in Fig. 8-30.

First, they can only be triggered on the high to low edge of the trigger pulse; it might be useful to have the option of triggering on either rising or falling edges.

Second, the circuit cannot be retriggered during the high output. Because of this, there is no way of further prolonging the output pulse by multiple trigger pulses.

Third, there is a *refractory period* after the positive Q output square wave that is proportional to the number of inverters; during this time the circuit cannot be triggered for a second pulse because the clear line is low.

Fourth, this device may be readily triggered by noise transients in the circuit in which it is placed; this is a serious drawback, as false triggering can make a given piece of equipment totally unreliable and of no use whatsoever.

Fifth, because of certain design considerations, there is an upper limit on the duration of the

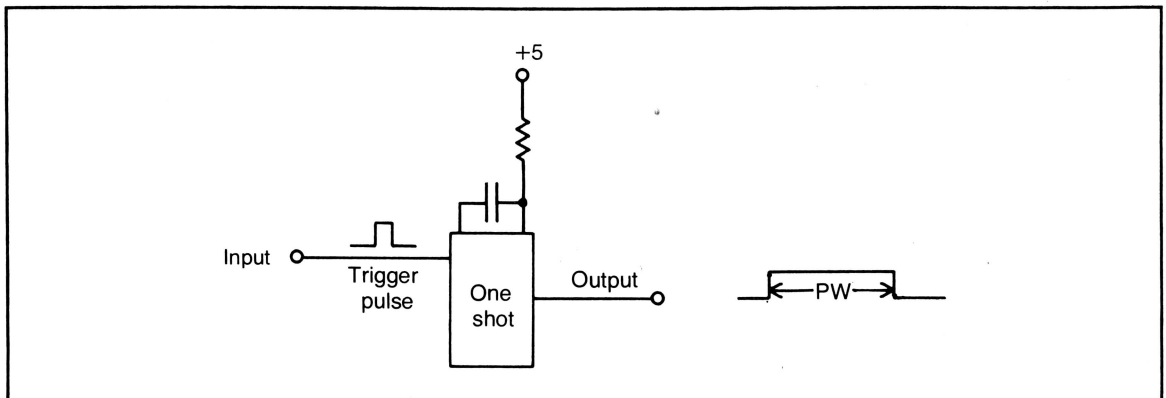


Fig. 8-27. Generalized one shot.

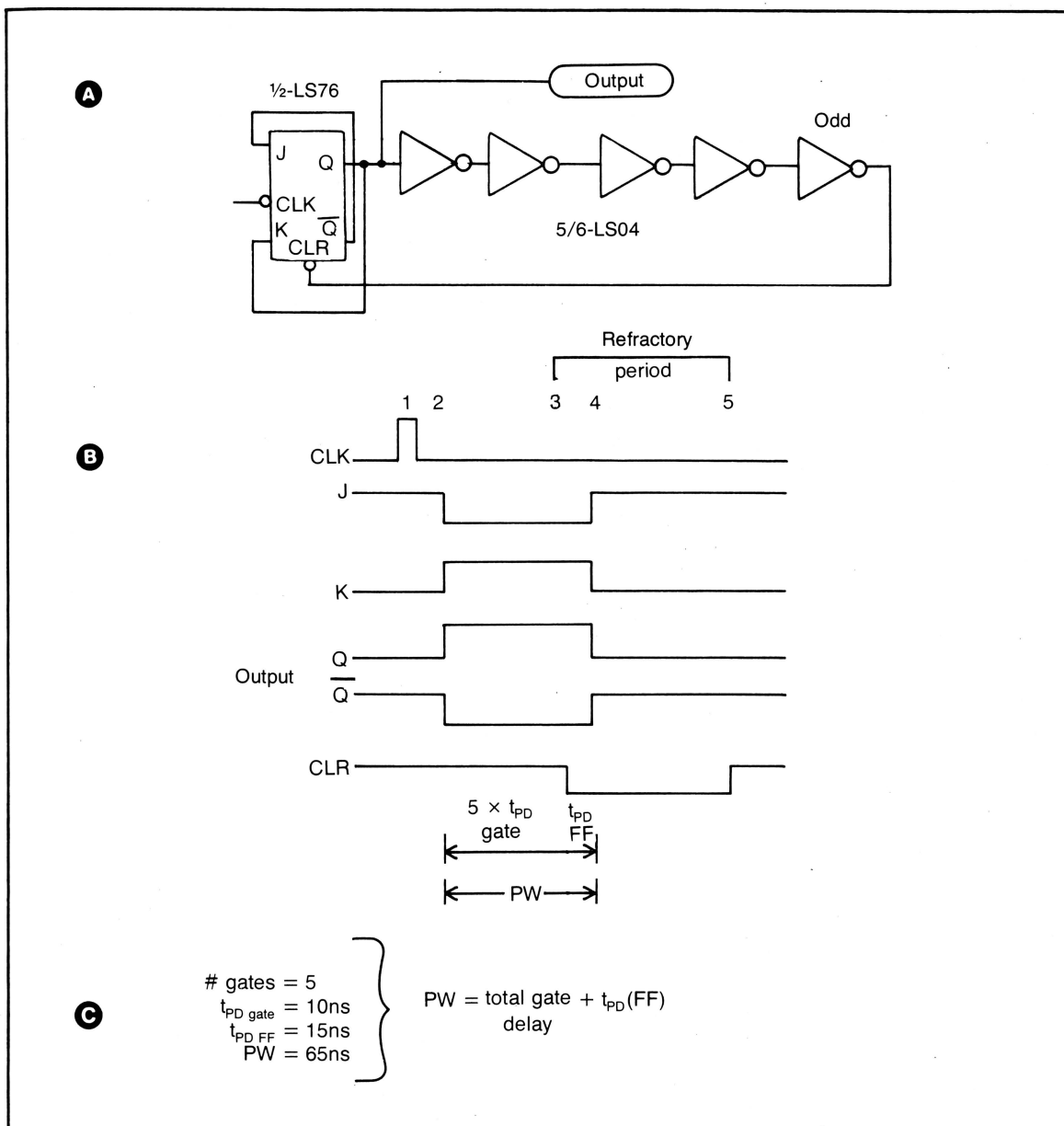


Fig. 8-28. Use of the J-K flip-flop and an inverted delay chain to create a one shot.

output pulse, on the order of a few hundred nanoseconds.

Finally, and perhaps as serious a drawback as any, there is the necessity for the extra components and additional connections for this rather limited set of functions.

Schmitt Trigger Inputs

As far as the noise problem (spurious triggering) is concerned, there is a solution: the Schmitt trigger input. You can appreciate what a Schmitt trigger does without going into the transistor-by-resistor details of the circuit. Basically, these cir-

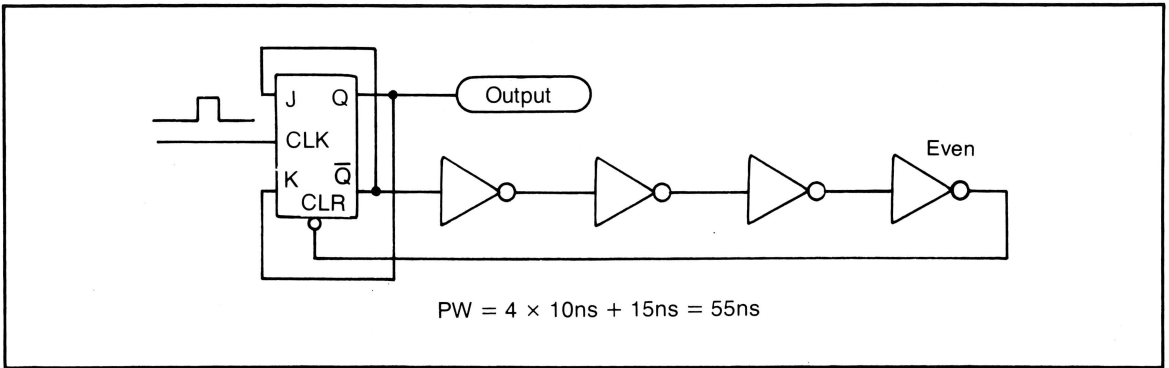


Fig. 8-29. Same as Fig. 8-28 with an even number of inverters.

cuits respond differently to rising and falling voltage levels and thereby improve noise immunity. Schmitt trigger inputs are incorporated in SSI gates such as the LS132 NAND and the LS14 inverter shown in Fig. 8-31A and into other more complex ICs. Let's see how the LS14 inverter handles a noisy input.

Suppose that what was once a neat square

wave has deteriorated into the sorry mess shown in Fig. 8-31B. This can happen through superposition of noise picked up in bus and transmission lines, and from passing through capacitive loads. A Schmitt trigger inverter will respond to a low to high transition of this signal only when the input voltage level reaches 1.6 volts. But, the Schmitt trigger will not respond to the falling edge of a waveform until it

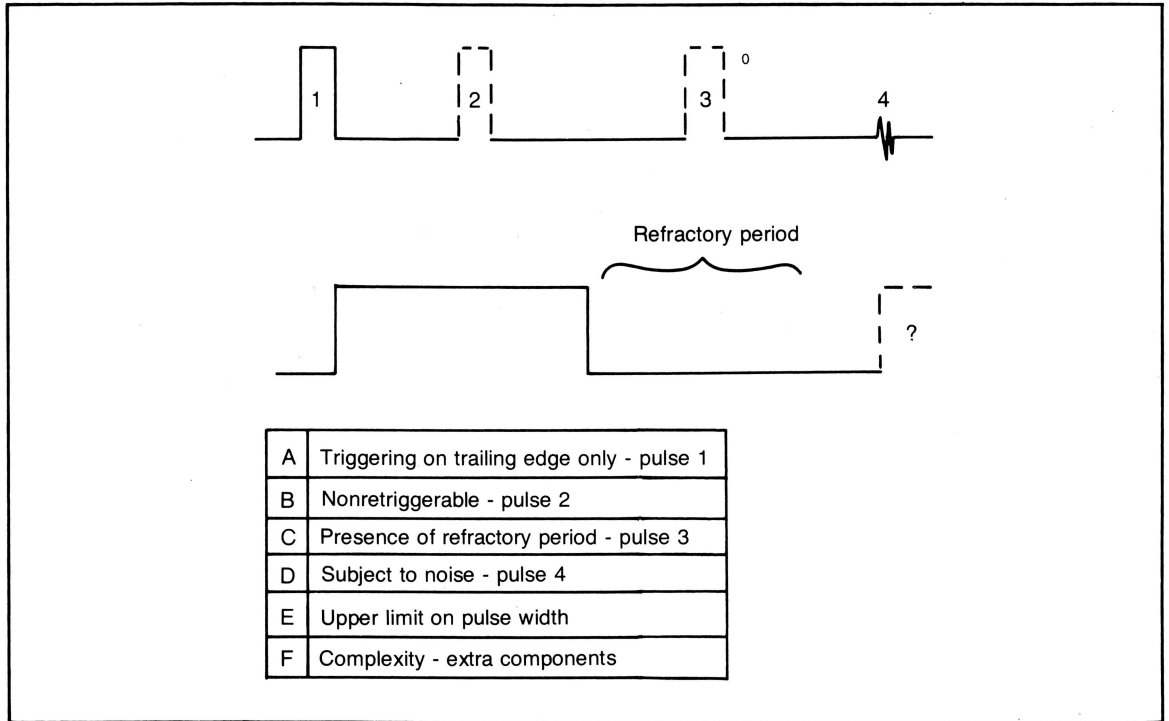


Fig. 8-30. Major drawbacks of the one shots just mentioned.

reaches 0.8 volts. These two levels are labeled as $+V_T$ and $-V_T$ in the figure, respectively. The difference between them, 0.8 volts, is called the *hysteresis* of the circuit. What purpose does this difference serve?

Let's see how this principle works to improve noise immunity. Assume you had designated the minimum value of a valid logic high input level as 2.0 volts, a typical TTL value. Then point A—the transient dip in Fig. 8-31B—would be seen as a brief, and erroneous, low. Likewise, we might specify 0.8 volts as the maximum value for an input

logic low. Then point B would be seen as a logic high instead of a low—another erroneous spike.

But instead, these two transients (which transgress normal TTL input logic levels) do not affect the input seen by the Schmitt trigger input circuit in the LS14 inverter. Thanks to the hysteresis—the difference between what is recognized as logic low or high depending on whether the signal is rising or falling—there is improved noise immunity. Noise peaks into this region (0.8 to 1.6 volts) from below or dips into it from above do not influence the device. As a result, the inverter input only

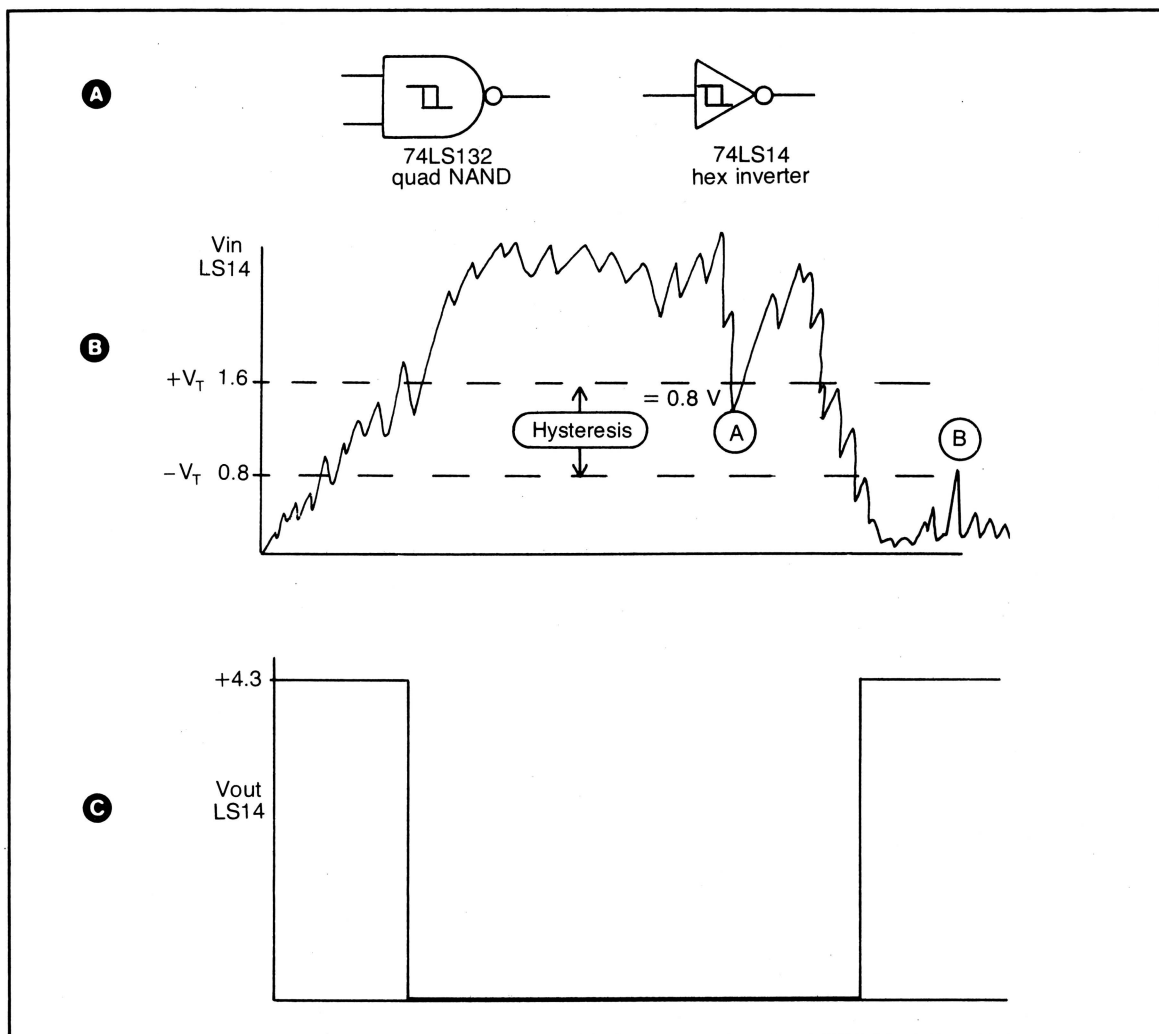


Fig. 8-31. Advantages of the Schmitt trigger: noise immunity and cleanup of sloppy signals.

sees a low when the signal from above crosses the 0.8 V level. Similarly, only when the signal from below rises above the 1.6 V level does the output revert to high. The neat square wave that results is shown in Fig. 8-31C.

Schmitt trigger circuits are used in the input circuits in many digital ICs. They are particularly useful in one shots and in counters where extraneous pulses could mistrigger the device. Sometimes the technical or data manual will not refer to a Schmitt trigger as part of the given IC. Rather, the literature may refer to a certain amount of “hysteresis built into the input for improved noise immunity,” or similar wording. In this example, the device has a hysteresis of 800 mV.

Typical One Shot Devices

The 74LS123 and the 558 quad timer are presented in the following section so that you can become familiar with their operation. These devices are extremely useful in digital projects.

The Dual Retriggerable 74LS123 One Shot. These are dedicated monostable integrated circuits that solve the other problems mentioned in the last section. One of these is the 74LS123 shown in Fig. 8-32. It has the following features.

- ☐ Hysteresis built into the inputs of the device. (Schmitt trigger-type operation.)

- ☐ Option of retriggering on rising or falling edge of the trigger pulse.

- ☐ Retriggerable, so that output pulse length can be extended by multiple trigger pulses.

- ☐ Essentially no refractory period.

- ☐ Improved noise immunity through input hysteresis, much like the Schmitt trigger circuit just discussed.

- ☐ Wide range of pulse widths—from tens of nanoseconds to many seconds.

- ☐ Relatively compact, only one resistor and capacitor needed to set timing parameters.

In addition, the LS123 is a dual-rail device, and has a clear input that sets the Q output to zero.

Note that there are two one shot devices per package, allowing you to create delay circuits with a

single chip, as you'll demonstrate later.

The LS123 is obviously a very versatile device. It is also not immediately understandable from mere inspection. The best way to approach the device is to look at its different modes of operation.

First, there are potentially three different trigger inputs—A, B and CLR—which are gated through the AND device, as you can see in the figure. However, since CLR is also connected to the clear input of the flip-flop, it is usually best to employ this line solely as a reset line, not as a trigger line. This is not emphasized enough in the data manuals, but will be dealt with below.

Basically, there are three types of input conditions for the LS123.

- ☐ A quiescent state, in which the output is low.

- ☐ An armed state, in which the device will “fire” a positive square wave pulse when triggered.

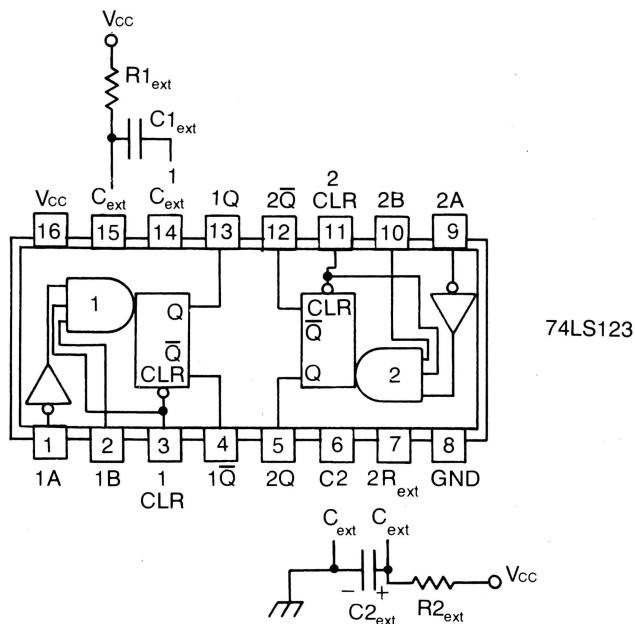
- ☐ A reset state, in which the clear input sets the output low, regardless of the other input conditions.

Figure 8-32B presents the state table as it commonly appears in the data manuals. There have been additions made to the margins of the table for purposes of clarification.

The conditions for output low are given in lines 1 to 3. The output of the AND gate must go high in order for the device to be triggered. Therefore, if any of the inputs to the AND gate (gate 1 in the figure, for instance) is low, then the device cannot be triggered. A high on input A, which is inverted, would prevent triggering—hence, it makes no difference what the values on B and CLR are, as denoted by the X or don't care symbol. The same reasoning applies to lines 2 and 3.

Assume for the moment that you want input A to be the trigger input. Then, B and CLR must be high in order to enable the device, and A must also be high in preparation for a down-going (triggering) signal. That is, if the device is to use input A as the trigger, the device is armed by setting A, B and CLR to high. Then, when A is brought low (and then high again), as indicated by the arrow in line 4, the

A



B

		Inputs			Outputs	
		A	B	CLR	Q	\bar{Q}
Output low (Q) conditions	①	H	X	X	L	H
	②	X	L	X	L	H
	③	X	X	L	L	H
Conditions for triggering	④	↓	H	H	↓	↑
	⑤	L	↑	H	↑	↓
	⑥	L	H	↑	↓	↑
Just trg'd by A	→ ⑦	H	X	↓	↓	↑
Just trg'd by B	→ ⑧	X	L	↓	↓	↑

Armed: see text

Reset

C

$$PW = 0.45 R_{ext} C_{ext} \quad \text{where} \quad \left\{ \begin{array}{l} PW \text{ in nanosec } (10^{-9}) \\ R \text{ in K ohms } (10^3) \\ C \text{ in pF } (10^{-12}) \end{array} \right. \quad \text{or} \quad \left\{ \begin{array}{l} PW \text{ in seconds} \\ R \text{ in ohms} \\ C \text{ in farads} \end{array} \right.$$

Fig. 8-32. Pin-out and state table of the LS123 integrated one shot, which employs Schmitt trigger type hysteresis at the inputs.

device will fire; Q goes high and Q' goes low for a predetermined interval. At any time during the output pulse, CLR can be brought low to terminate the output high by placing a low on CLR and bringing it high again, as given in line 7. It is important that A will have returned to high before you attempt to clear the device, however. (Stop and try to answer why this is so.)

Likewise, to use B as the trigger input, set A low and CLR high. Leave B low, so that you are assured that Q is low, just before triggering. By bringing B high (and then quickly low again), the device is triggered. To terminate the high output on Q, CLR is again pulsed low then high, as indicated in line 8. Again, B should already have returned to low before you perform this clear. The reason, if you've not already guessed, is as follows.

If you bring CLR low then high with B still high (remember A is still low in order to use B as a trigger) then the device will retrigger! This is the same as the situation in line 6. Similarly, if you were using A as trigger, then your failure to bring A high again would again result in the situation given in line 6: the rising edge on CLR would retrigger the device!

In short, the triggering input must return to its inactive level (high for B and CLR, low for A), immediately after triggering the device.

Regarding the use of external resistors and capacitances in the LS123, observe a few simple rules. First, the formula for pulse width PW is given by

$$PW = 0.45 R_{\text{ext}} \times C_{\text{ext}}$$

where PW is in seconds.

R is in ohms.

C is in farads.

The external resistor should not fall outside of the value range of 10 to 200 K for reliable operation. (Some data manuals allow a bit more leeway.) Theoretically, there is no restriction on the value of the external capacitor. However, for small values—below 1,000 pf or .001 μF —the equation of PW is a bit inaccurate, and a graph supplied in the data manuals has to be used. In general, for pulse

lengths of a few microseconds or more, the formula is reliable.

If using large values of capacitance to obtain long (several seconds) output pulses, electrolytics are necessary. When using them, make sure the polarity of the connections are as shown in Fig. 8-32A, with the negative end grounded.

Finally then, we can summarize the proper use of the LS123 as follows.

□ If you want to employ a falling edge as the trigger, use input A. All three inputs should be high in order to arm the device. A very short low pulse (low then back to high) on A will trigger the flip-flop. The output can be terminated at any time with another short negative pulse on CLR. If A has not returned to high, however, the clear pulse may retrigger the device, for the reasons mentioned above. (Condition in line 6 of the state table.) The valid sequence of conditions with A as the trigger corresponds to lines 1, 4, and 7 of the state table of Fig. 8-32B.

□ If you want to employ a rising edge as the trigger, use input B. Both B and CLR should be high, and A low in order to arm the device. A short positive pulse on B triggers the one shot. A short down-going pulse on CLR terminates the output. Again, if B has not returned to low when the clear pulse occurs, you will retrigger the device. The valid sequence with B as the trigger corresponds to lines 2, 5, and 8 in the state table.

□ Use the formula for pulse width given above in order to determine the values of the external resistor and capacitor. Make sure you ground the negative end of electrolytics if they are used.

□ The timing diagram of Fig. 8-33 gives the sequence for A as trigger. The diagram shows the conditions for arming, triggering, retriggering (point x), clearing (point y) and rearming (point z). The three output waves generated are labeled as A, B and C, respectively. The encircled numbers at the top of the diagram correspond to the line numbers in the state table. A timing diagram corresponding to points 2, 5, and 8 for B as trigger would be essentially the same, except for the polarity of B.

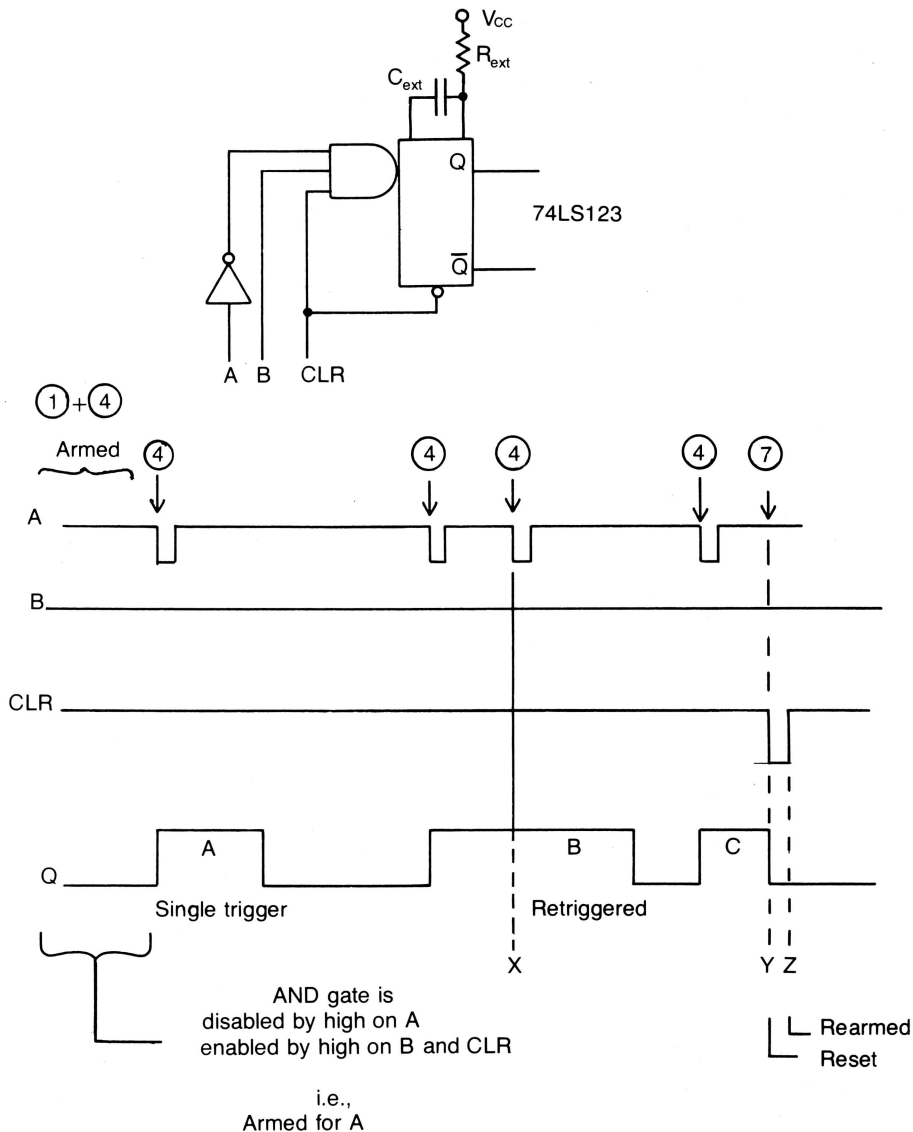


Fig. 8-33. Another representation of the function of the LS123, using the example of input A as the trigger input.

The 558 Quad Timer. The LS123 may seem a rather complex device at first exposure. If you can get by with less flexibility, a simpler one shot may be appropriate. The 558 quad timer is a dedicated monostable with four one shots in one package. Strictly speaking, it is a hybrid linear/digital device

which consists of so-called operational amplifiers and flip-flops as the chief functional active components. Again, our concern is with the overall operation of the device, rather than the details of internal circuitry.

Figure 8-34 illustrates the 558 quad timer. In

Fig. 8-34A is the labeled pin-out of the entire package. In Fig. 8-34B is one of the timers. The features of this device are as follows:

- ☐ A trigger input TR, which is triggered by a negative-going edge.
- ☐ A reset input, active low, which overrides TR. Note that the reset (pin 13) is connected to all RES inputs internally. A negative pulse on this pin brings all outputs to zero.
- ☐ A pin for setting the duration of the output pulse. The external resistor and capacitor are attached to this timing pin, TM, as shown in the figure.
- ☐ A single output, producing a positive square wave pulse when the device is triggered.

The formula for determining the pulse width is

$$PW = R \times C$$

where PW is in seconds.

R is in ohms.

C is in farads.

The 558 is used in the game port circuitry in the Apple for analog inputs. This is the device which produces a pulse with the duration proportional to an external resistance. The value of the resistance is in turn determined by the position of a joystick or game paddle.

Applications for One Shots

The first application is as a pulse stretcher, or

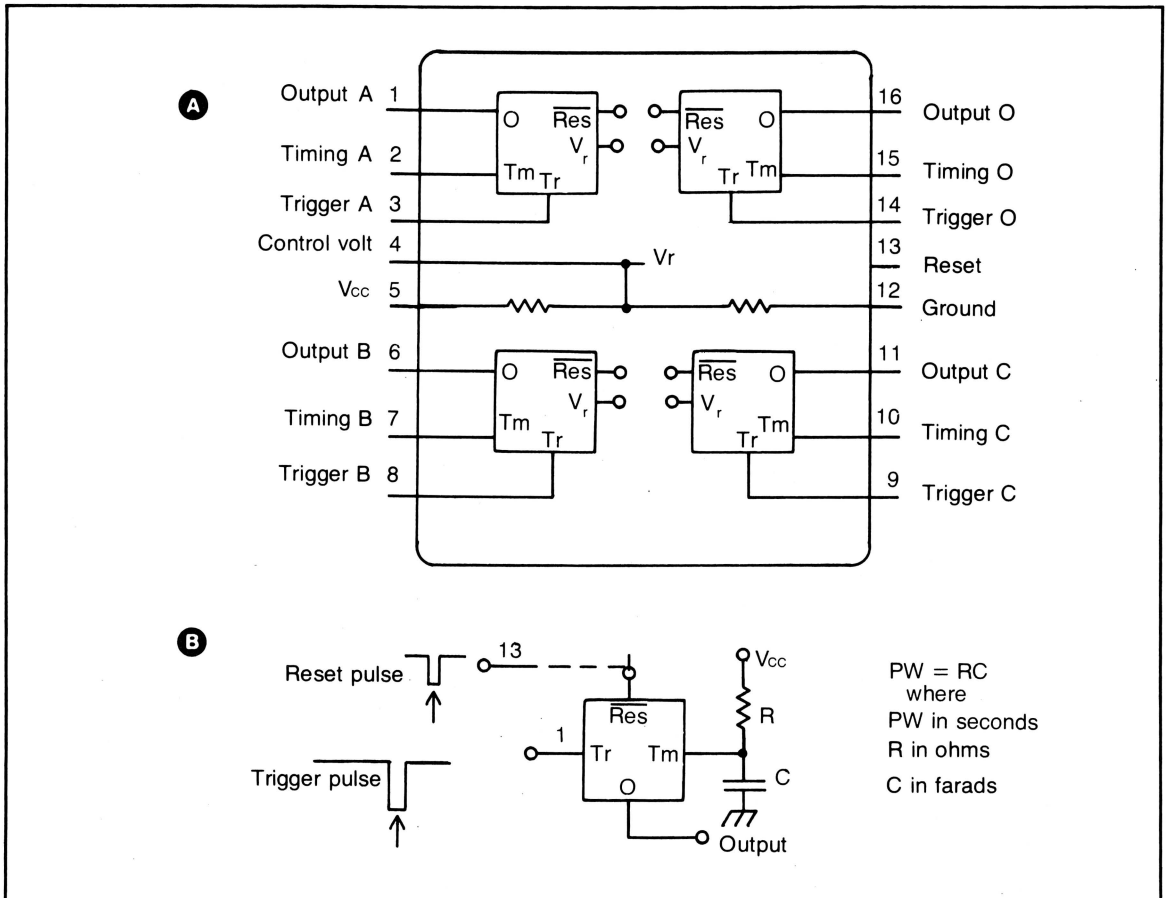


Fig. 8-34. The 558 quad timer used in the Apple for game paddle reading.

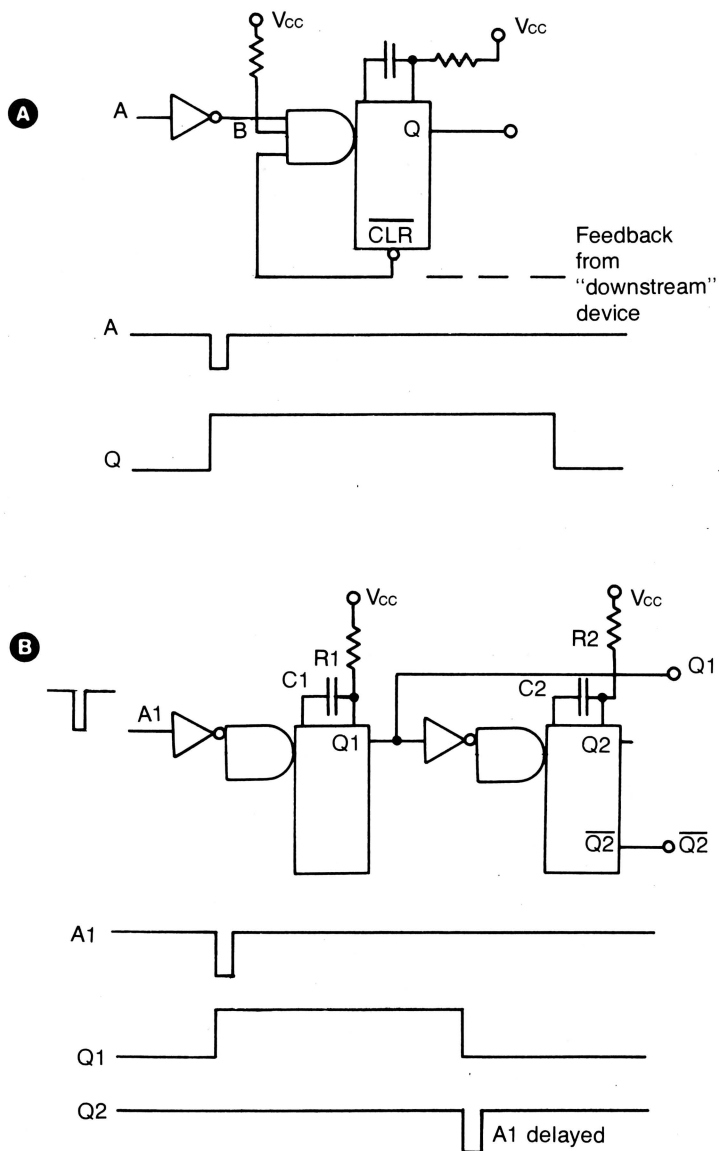


Fig. 8-35. Applications of the one shot as pulse stretcher and as delay line.

single pulse generator. This is illustrated in Fig. 8-35A. A short pulse produces a positive going square wave of fixed duration. The configuration for the LS123, using input A, is shown.

Another application is as a *delay line*. You'll remember that a chain of inverters could delay a

pulse by a brief interval (tens of nanoseconds). For longer delays, the configuration in Fig. 8-35B may be used. The short input pulse on line A1 can be delayed for some set interval by choosing the values of R1 and C1 appropriately (delay = $0.45 R1 \times C1$). The values of R2 and C2 must be chosen

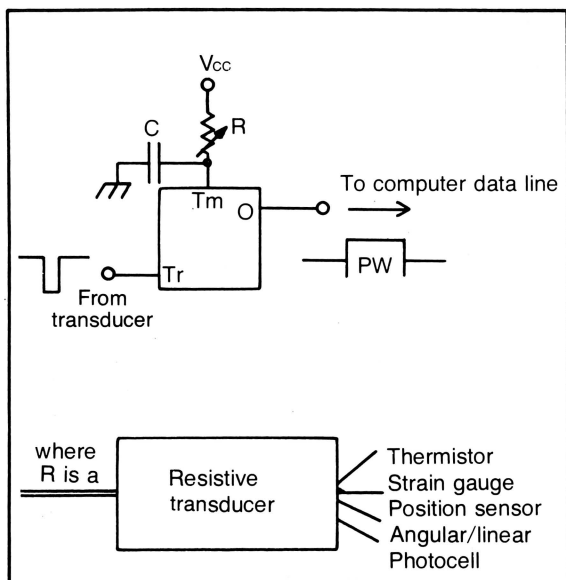


Fig. 8-36. Application of the one shot as a key element in A/D conversion circuits that employ resistive transducers.

to produce an output from the second one shot with the duration equal to that of the original input pulse, A1. Since A1 was negative, the output is taken off of Q2', the complementary output.

Last, an obviously useful application of the one shot is as a measurement device. Specifically, the one shot can be used with *resistive transducers*, that is, passive components whose resistance varies in proportion to some physical quantity. The physical parameter may be heat, light, mechanical stress, position, etc. This is an uncomplicated method of analog to digital (A/D) conversion which can be employed in a wide variety of applications. The variable resistance could be the potentiometer in a game paddle, or in a resistive photocell, strain gauge, etc. Even chemical quantities such as pH can be measured by means of this basic circuit.

Naturally, appropriate software must be written to measure the actual length of the output pulse produced by this single bit A/D converter. The

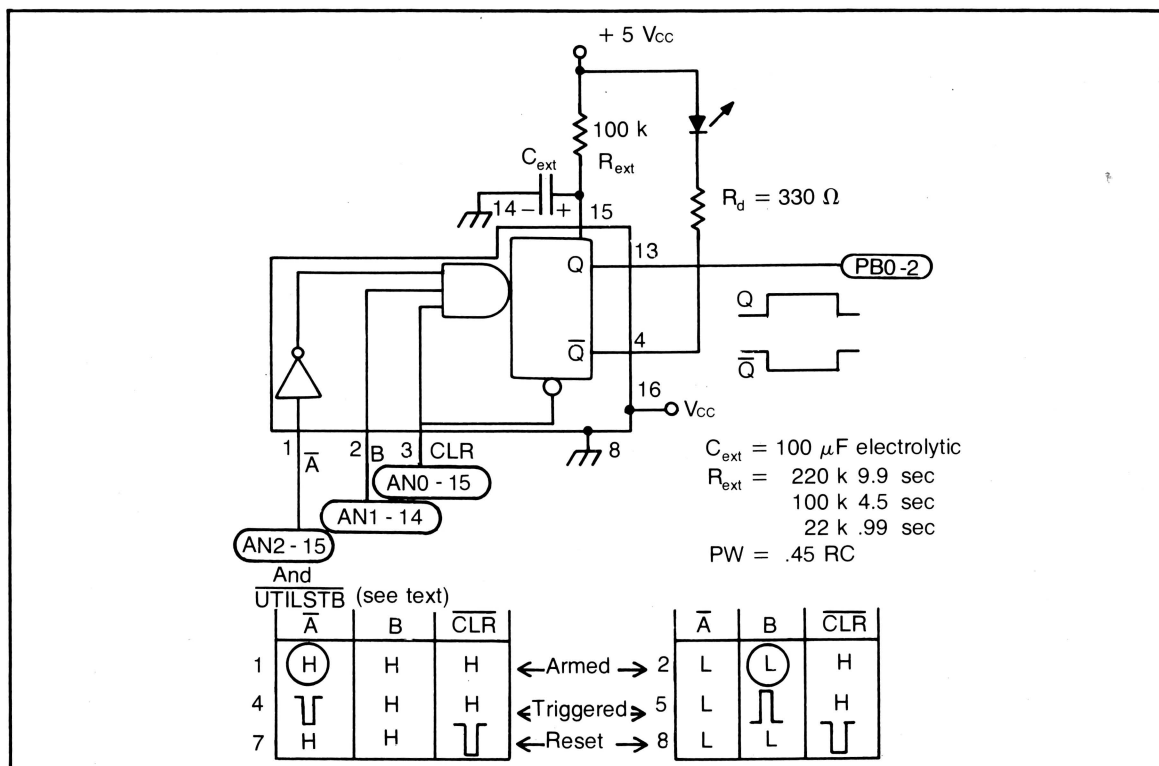


Fig. 8-37. Experiment 15. Basic operation of the LS123.

program is best written in machine language for speed. The first task of such a program is to reset the one shot to zero, and then begin a timing loop. On each passage through the loop the program will check to see if the value of the output is high or low. The loop also increments a memory location by one on each cycle. When the value goes low, the timing routine stops counting. The value in the memory cell will then be proportional to the resistance, and hence to the physical quantity measured. Further processing for numerical and even graphical output may then be performed after the measurement.

We'll explore this application in the last chapter.

EXPERIMENT 15, THE ONE SHOT

Purpose

To illustrate the operation of a typical one shot, the LS123, along with one of its applications.

Materials

- 1 - 74LS123 dual retriggerable one-shot
- 2 - 330 ohm resistors
- 1 - 22 K resistor
- 2 - 100 K resistors
- 1 - 22 K resistor
- 1 - 10 μ F electrolytic capacitor
- 2 - 100 μ F electrolytic capacitors

Procedure

1. Connect the circuit in Fig. 8-37. Use an R_{ext} of 100 K to start with. The LED adds a little interest. To examine the various modes of operation, first set up an appropriate heading on BDIS and set all input lines high. The device may be triggered during this process. Now, leaving A high, toggle the other two lines, as suggested in Table 8-8A. Then, setting B and CLR high, toggle A low then high quickly, as indicated in Table 8-8B. Time the output pulse. Do this several times.

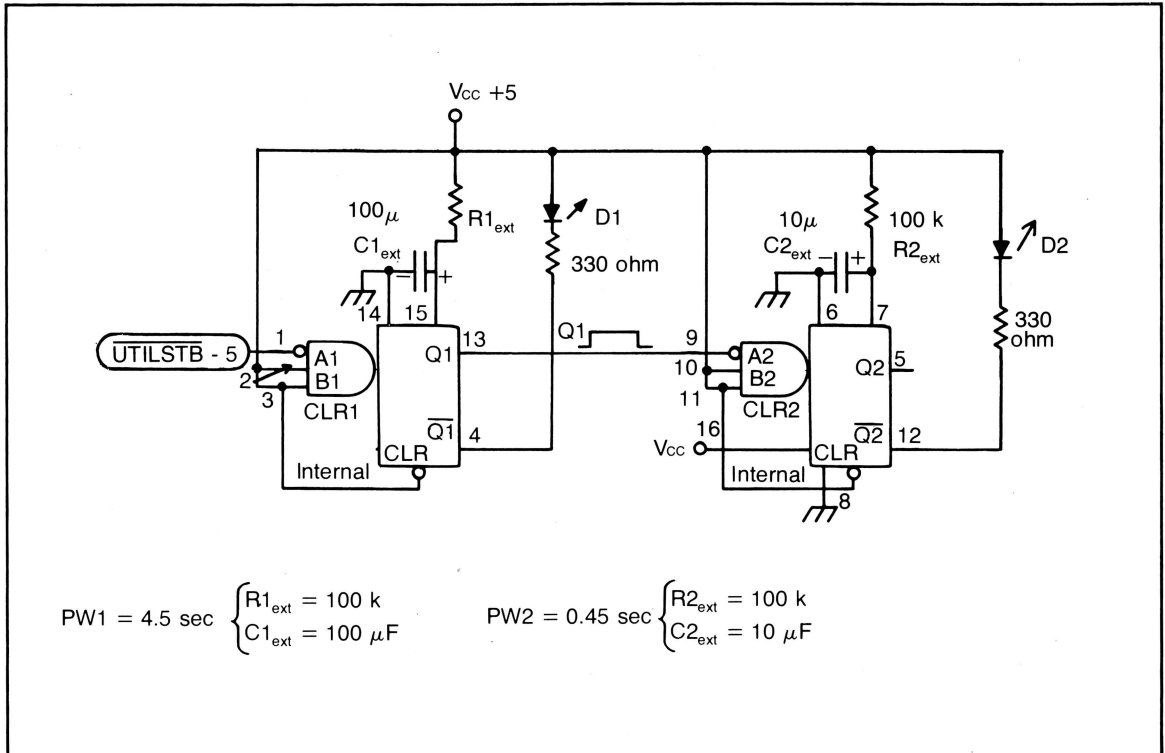


Fig. 8-38. Experiment 15. Configuration of an LS123 package as a delay line.

2. Try retrigging the device before the output pulse goes low. Try clearing the device before the pulse ends. Then try substituting various resistance values, such as 220 K or 22 K and repeat the trigger, retrigger and clear actions.

3. As an option, you may want to try using the B input, which triggers the device on a low to high transition. The procedure is similar to that suggested in steps 1 and 2 above, with the screen display indicated in Table 8-8C and D. Note that the use of A and B inputs will correspond to the tables appearing below the circuit schematic given in Fig. 8-37.

4. Now hook up the circuit in Fig. 8-38. Using different colored LEDs if you wish. Note that here we are using the utility strobe directly, as this is a very short ($\frac{1}{2} \mu\text{sec}$) negative-going pulse, ideally suited as an input signal to the A trigger. What function does the circuit perform?

Discussion

After taking the one shot through its paces in steps 1 to 3, you examined a delay line, or pulse delay, circuit in step 4. The output was taken off $\overline{Q}2$, so the delayed pulse was in fact inverted. It was also considerably longer than $\frac{1}{2}$ microseconds, for purposes of demonstration.

You had the option of using different resistances in the one shot demo. Did you think of substituting different capacitance values, or of using, say, a 100 K pot to continuously vary the pulse length?

One other application which is probably now apparent, is that the one shot could be used as a basis for rough measurements of capacitance and resistance, much as you would use a piece of test equipment such as an ohmmeter. If computer based, creating the software would probably consume the lion's share of project time. You would have to worry about a number of factors, such as the range of measurement, how long it would take to measure very large values of resistance and capacitance and, of course, how accurate the final measurements would be. The Apple has its own one

Table 8-8. Experiment 15 (A and B Using Input A in the LS123 as the Trigger. C and D Using Input B as the Trigger.)

A	GPSIG:	AN2	AN1	AN0:	PB0
	GPIN#:	13	14	15:	2
	LABL1:	A'	B	CL!:	Q
	LABL2:	1	2	3:	13

	0	'1	1/0	1/0:	0

B	GPSIG:	AN2	AN1	AN0:	PB0
	GPIN#:	13	14	15:	2
	LABL1:	A'	B	CL!:	Q
	LABL2:	1	2	3:	13

	0	'1/0	1	1:	0

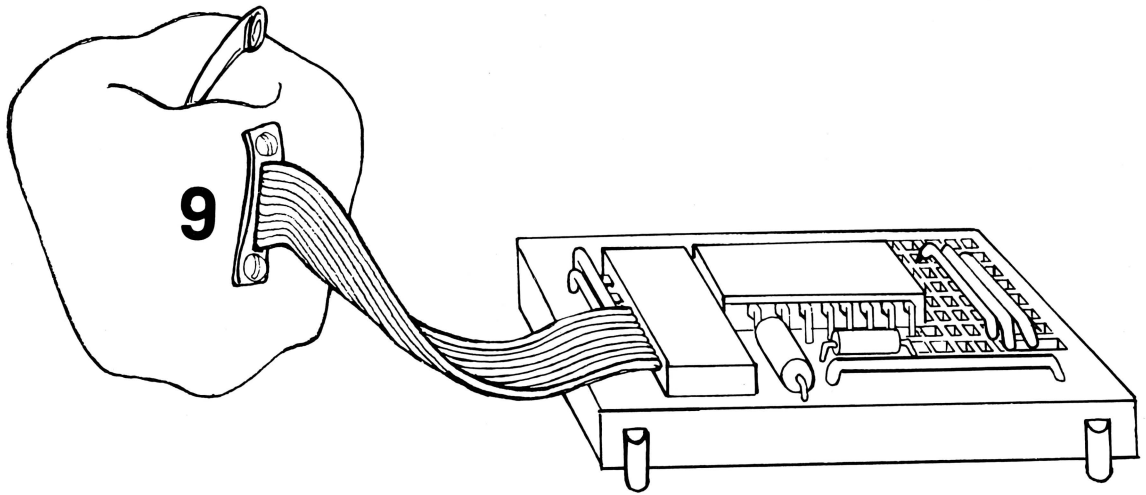
C	GPSIG:	AN2	AN1	AN0:	PB0
	GPIN#:	13	14	15:	2
	LABL1:	A'	B	CL!:	Q
	LABL2:	1	2	3:	13

	0	0/1	'0	1/0:	0

D	GPSIG:	AN2	AN1	AN0:	PB0
	GPIN#:	13	14	15:	2
	LABL1:	A'	B	CL!:	Q
	LABL2:	1	2	3:	13

	0	0	'0/1	1:	0

shot circuitry and monitor software for reading it already built in. Again, this is a subject to be explored later on.



MSI Sequential Devices

Medium scale integration (MSI) logic provides a broad and diverse middle range in digital device complexity.

By convention, MSI encompasses devices that are composed of more than 12 but less than 100 gate equivalents per device. MSI logic provides the user with tremendous gains in both economy and ease of design. Instead of having to string together many SSI devices to perform a basic task, the user can select from a wide variety of digital functions—both sequential and combinational—in a prepackaged form. Often, the user can replace a multipackage SSI circuit with a single MSI package directly. He enjoys not only the savings in design time, but often realizes additional functions with the MSI chip that are unavailable with the discrete design using SSI components. Many MSI functions are implemented on a convenient 16-pin DIP package, and this results in further savings of space and often in power consumption as well.

In this and the next chapter, you'll learn about the major combinational and sequential MSI func-

tions by examining representative chips from each group by experiment. First, let's look briefly at the advantages and major functional categories of MSI devices.

MEDIUM SCALE INTEGRATION

When compared to the SSI level of complexity, the advantages of MSI logic can be summarized in one phrase: reduced overhead. The four-bit binary ripple counter that you constructed in Experiment 15 would need the following in a permanent installation:

- ☐ 2 74LS76 chips.
- ☐ 2 16-pin IC sockets (optional but desirable).
- ☐ Board space for the two devices.
- ☐ Power and ground lines for both packages.
- ☐ About 8 mA of current (I_{cc}).
- ☐ Time/labor/cost of mounting and connecting two packages.

You can replace the two-chip LS76 circuit with a single MSI chip, the 74LS93 counter. This device is a four-bit binary ripple counter which performs the same function as the two-chip SSI circuit. (Refer to your TTL data manual for details on the LS93). The LS93 counter has the additional feature of gated clear inputs as well. The circuitry is contained on a single 14-pin DIP package. Current consumption is 9 mA, which is about the same as two LS76 packages.

If you need only a four-bit binary counter in your applications circuit, the LS93 is an ideal replacement for the two-chip SSI circuit. You use HALF the board space and mounting hardware. Fewer power and ground connections are necessary, and you are handling only one IC instead of two. Since you're using one IC, no inter-package wiring is necessary. Finally, the cost is less. For instance, typical wholesale/discount prices are about \$.80 for the two LS76 ICs and only \$.55 for the single LS93, a savings of over 30%.

There is another less obvious advantage to substituting a single MSI chip for a multipackage SSI circuit: improved reliability.

The failure rate of an IC package is expressed as the *mean time before failure*, or MTBF. The MTBF refers to the entire package, not to the individual gates of flip-flops on the package. Within broad limits, the failure rate of a digital circuit as a whole is a function of the number of packages that it contains. Therefore, the MTBF for the one-chip MSI counter will be longer than that of the two-chip SSI circuit, and hence the circuit is more reliable.

Now extend the above idea of replacing SSI circuits with prepackaged MSI functions. Overhead (mounting, labor, space, etc.) is reduced even further. Assume you're designing a small digital system or module. Your application circuit consists of two small PC (printed circuit) boards with 20 SSI packages on each, for a total of 40 chips. The overhead of materials—sockets, boards, board mounting, wire harnessing between boards—as well as the labor involved, makes up the bulk of system cost.

A reasonable estimate is that you would spend twice as much on nonchip hardware and labor as you

would on the IC's themselves. Let's say the chips cost \$10 and the labor and nonchip overhead costs \$20. The ratio is usually a lot higher. One reason is that just the sockets alone may be a fair percentage of the cost of the chips themselves. Chips can be directly soldered to the board, and sometimes this is done to reduce the outlay. However, if you don't use sockets, good luck when it becomes necessary to service the module or to replace an IC package.

Assume now that you replace the system with an MSI design in which, to be conservative, you enjoy only a two-to-one chip reduction. You'll be handling and mounting 20 fewer chips. You'll need one less board and 20 fewer IC sockets. There'll be less interpackage wiring and no necessity for wiring between boards. Also, fewer power and ground connections will be required. Even if the MSI chips cost twice as much (unlikely) as the SSI chips of the two-board circuit, you are still way ahead of the game in terms of materials and labor for nonchip components.

Half the number of chips improves the MTBF (mean time before failure). Another reason for the improvement in this reliability factor is the fewer electrical connections required.

At the very least, you will have cut the more costly and time consuming part of circuit construction—nonchip overhead—in half. For this example, you would have saved \$10 in overhead, for a total savings of better than 30%. And you would have a smaller and more reliable package in the bargain.

As the system gets larger and more complex, overhead increases, and so the advantages of using MSI increase. This is because of the need to mechanically mount boards in subassemblies, to make connections between boards and to distribute power among them. Also, the usual replacement ratio is close to three or four SSI packages per MSI package. Therefore, as a general rule, in larger systems the chip to nonchip overhead approaches 80 to 90% of total system cost.

Finally, there is greater ease of design with MSI. Because of the range of available MSI functions, and the flexibility built into each device, design can be *structured*. That is, you can specify

functional blocks of modules within the circuit, much as you would when roughing out a program in BASIC. Quite often, you can then select MSI packages which correspond to these circuit functions.

SSI devices do have a role in this scheme, however. There are certain minor functions within any circuit which do not require the power and flexibility of an entire MSI package. Examples are buffering, gating, storing or inversion of one or two signals. A few (combinational) gates or (sequential) flip-flops are often necessary for these limited tasks. The role of SSI logic may be thought of as essentially that of a glue which holds the major MSI elements together.

In summary, even small circuits benefit from MSI replacement of SSI components because, even though you may save little or no money on the chips, you will save on nonchip hardware and you will reduce the time and effort spent in construction and check out of the circuit. Circuit reliability is also improved due to reduced chip count and fewer electrical connections.

Further, with substitution ratios greater than two to one, you will realize savings in power supply requirements as well. Larger circuits benefit even more from MSI substitution. Also, designing larger circuits and systems is much faster and easier with MSI logic.

System design is certainly beyond the scope of this book. But you will get an idea of how simple MSI circuits are constructed and how SSI elements can be introduced to augment a basic MSI package in a few of the experiments in this and the next chapter.

MSI Functions

Although LSI and VLSI devices play a critical role in more complex digital systems—as exemplified by peripheral controllers, microprocessors and computer memory—MSI devices nonetheless remain important. MSI logic performs the workhorse functions of a system.

MSI is employed in data display circuitry; in address decoding; in transferring data to and from address, data, and control buses in computers; in I/O applications for latching data; in routine count-

ing and timing operations; and in a host of similar tasks.

Important too is that your understanding of MSI will allow you to approach the complex LSI and VLSI functions more intelligently as you pursue your studies later on.

As just mentioned, there is a close analogy between designing with digital devices and programming in a computer language. An SSI device is like an individual command. It is not very meaningful in itself. It must be connected to other devices before it can do anything useful. MSI devices, on the other hand, can almost stand alone: they can perform a useful function without the need for additional circuitry. A decoder, adder, counter or register can be set up in a straightforward manner because the respective function is prepackaged in the device. MSI elements are, then, like little, self-contained subroutines. Hook enough of them together in the right order (with a little SSI “glue”) and you have a working system. This is roughly like programming, in which a final program is structured around several modules or subroutines.

Over the years, several MSI packages have gained pre-eminence because they incorporate the mix of features that give them flexibility in a variety of applications.

Some of the more popular MSI chips are listed in Table 9-1, under their functional heading. The two major categories are, of course, sequential and combinational.

Sequential MSI Logic. Within the sequential MSI category are the main classes of counters and registers.

Counters, typically four bits wide, can be of the synchronous or ripple variety. In synchronous counters all flip-flops change simultaneously with the clock pulse, while in ripple counters the clock pulse cascades from one flip-flop to another. Synchronous counters are preferred, but they are somewhat more complex than their ripple counterparts. MSI counters are usually of the binary or BCD type. Some counters are quite flexible: they can count either up or down, perform a modulo-divide (divide-by-N) function, and can be preset to some 4-bit number as desired. The LS190/191 are

Table 9-1. Major MSI Combinational and Sequential Functions.

Sequential	
Counters	
Ripple Counters	LS93, LS197 (binary) LS90, LS196 (BCD)
Synchronous Up/Down Counters	LS193, LS191 (binary) LS192, LS190 (BCD)
Shift Registers	
4-bits wide	LS194 (bidirectional, MSI) LS195 (shift right only, MSI)
8-bits wide	LS299 (LSI, universal, 3-state outputs)
16-bits wide	LS670 (an LSI "register file")
Memory	
8-bit latch	LS373 (MSI transparent 3-state latch)
64-bit RAM	LS189 (LSI RAM with 16 4-bit words)
256-byte PROM	LS371 (LSI PROM with 3-state outputs)
Combinational	
Decoders	
Encoders	
Multiplexers/Data Selectors	
Demultiplexers/Data Distributors	
XOR (exclusive-OR) Functions	
Three-state Buffers/Transceivers	
Arithmetic and Logic Units/Arithmetic Processors	

Note: The first six combinational groups are covered in Chapter 10. The last combinational category is composed mostly of LSI logic, and usually consists of devices with both combinational and sequential elements.

examples. Other counters can act as transparent latches as well as counters. Some have asynchronous clear lines, which set all bits to zero, independent of the state of the other inputs.

MSI registers, like some counters, can hold and store binary data. They too are usually four bits wide. However, the registers I refer to—known as shift registers—do more than simply latch and hold data; they are dynamic storage devices. Once data is loaded in serial or parallel form, the data can be shifted one bit at a time, often bidirectionally. Shift registers form the basis for parallel-to-serial and serial-to-parallel conversion. You'll note in the table that larger, full-featured shift universal registers—the 8-bit, bidirectional LS670 with three-state output, for instance—fall into the LSI range of complexity. The principles of operation of such devices are quite similar to their smaller MSI brothers.

Last, there is a third class of sequential functions mentioned in the table: memory devices. There is no qualitative difference between memory components and latches, as they both store data in some multibit format. It is just that memory devices contain many more storage elements (flip-flops) than the so-called latches. Almost all memory falls into the LSI range of 100 or more gate equivalents per device. Even a small 64-bit RAM (random access or read/write memory) such as the LS189 is an LSI component. It is organized into sixteen 4-bit words. Larger memory devices like the 256 byte (8 bits per word) LS371 PROM (programmable read only memory) are well into the LSI range. We will not be dealing with LSI memory in this chapter. However, you should note that the principles of operation of large scale memory are qualitatively similar to the smaller MSI sequential devices. So, you should have no great difficulty if and when the

time comes for you to design with memory devices.

Combinational MSI Devices. Also listed in Table 9-1 are the main combinational MSI functions. Seven categories are listed. Most of the arithmetic processors or ALUs (arithmetic and logic units) in the seventh category fall into the LSI class, as you would expect from the complexity implicit in the functions that they perform. Our concern will be with the first six groups of combinational MSI functions; these will be covered with experiments in the next chapter.

Note that the seventh category of ALU functions may be comprised of hybrid devices with both combinational and sequential elements on the same package. In fact, almost all of the LSI and VLSI devices you will encounter have both combinational and sequential components. In this sense, the neat division between combinational and sequential breaks down at higher levels of integration.

RIPPLE COUNTERS

The first of the MSI counters we'll examine is a 4-bit binary counter. An example of such a device is the 74LS197, which is illustrated in Figs. 9-1 and 9-2. It consists of four negative edge-triggered flip-flops, with appropriate gating circuitry. Though simple, this counter has a number of useful features.

- It can count from binary 0000 to 1111 (decimal 0 to 15). It can also serve as a divide-by 2, -4, -8, or -16 device.

- Alternatively, data can be loaded independently through four data input lines.

- There is an independent (asynchronous) clear which overrides all other inputs.

The design of the LS197 is much like that of the counter you built in the last chapter from two LS76 chips. When the external jumper is connected between output Q_A and the clock input of flip-flop B, the LS197 is set up as a full four bit counter.

As you can see from the schematic, the LS197 is a ripple counter: the clock signal ripples through the circuitry from one flip-flop to the next. Because the clock signal is cascaded, the downstream flip-

flops cannot change state until the prior flip-flop has done so. As a result, the outputs will not change simultaneously. Because of the inherent propagation delay, any change in the output state of each successive flip-flop is delayed by a small but finite interval. This is shown in Fig. 9-1 on the down-going edge of pulse 4 off output Q_A . This edge represents the transition from 0111 to 1000, where all flip-flops are changing state. As you can see, the propagation delay will result in a staggered transition, or ripple, of each respective flip-flop output, Q_A through Q_D .

With the external connection indicated, the LS197 serves to divide the input clock pulse (fed into clock 1) by 2, 4, 8, or 16, depending on which flip-flop is used as the final output. For instance, eight clock pulses must occur before pulse 4 off of Q_A is generated. At this same time, shown by the dashed vertical line in the timing diagram, Q_B will have finished its second pulse, Q_C only its first pulse, and Q_D will have just begun the leading edge of its first pulse.

Without the external connection, the device can act either as a divide-by-2, or as a divide-by-8 counter, using Q_A and Q_D respectively as the outputs.

Any logic low/0 signal on the clear line will reset all outputs to zero, regardless of the current inputs, or of the current state of the individual flip-flops. Clear is, then, an asynchronous input, in the sense of the term discussed in the last chapter.

The pin-out and state table for this counter are given in Fig. 9-2. The LS197 is a 14-pin IC package, which consumes 16 mA of supply current when unloaded. The inputs are clear, count/load, clock1 and clock2, and four data lines. Clock1 is the line used when the device is used as a 4-bit counter, with the external connection made between pins 5 and 6.

The state table merely presents the same information as the timing diagram and presumes 4-bit operation. Again, the device has three basic modes: count, load, and clear. To count, the count/load line must be high; to load data, it must be low. When counting, the output transitions occur on the down-going edge of the clock. When in loading

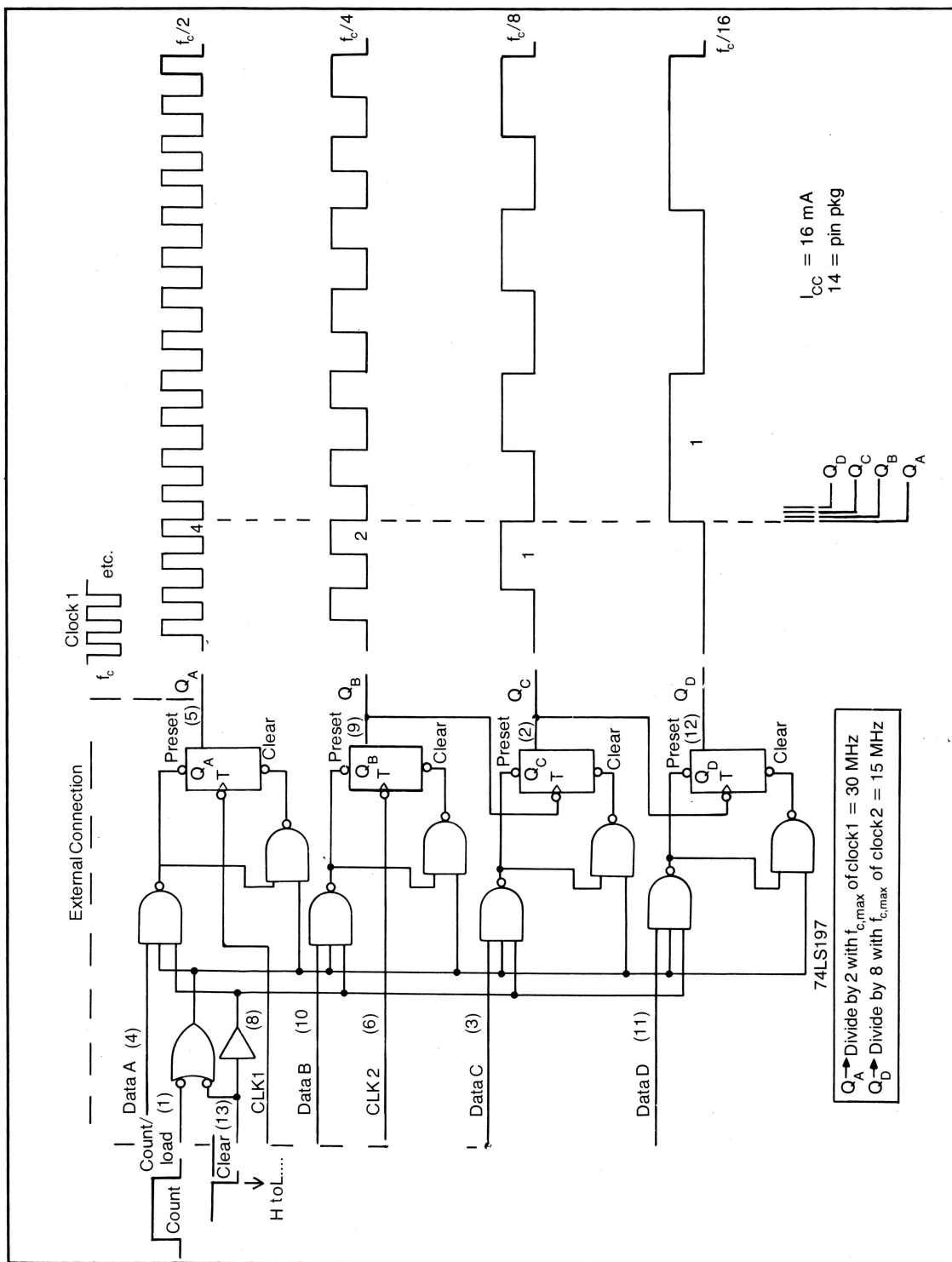
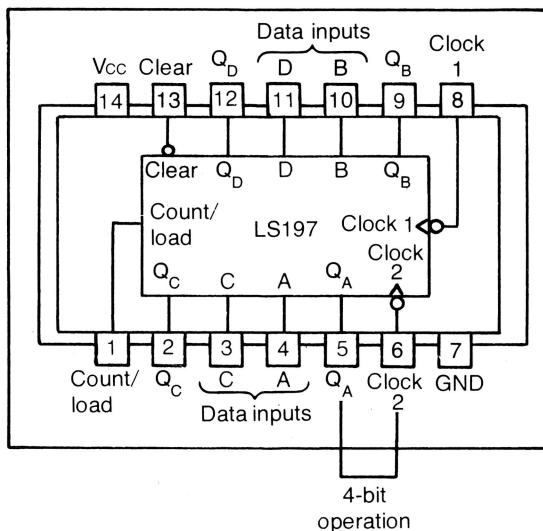


Fig. 9-1. The LS197 binary ripple counter set up as a four bit device.

A



Binary Cntr
74LS197



$I_{CC} = 16 \text{ mA}$
14 = pin

B

As 4-bit cntr:

CLR	Clock1	Count/ Load	Outputs Q_D Q_C Q_B Q_A				Decimal Equiv.
High	↓	High Count	L	L	L	L	0
			L	L	L	H	1
			L	L	H	L	2
			L	L	H	H	3
			L	H	L	L	4
			L	H	L	H	5
			L	H	H	L	6
			L	H	H	H	7
			H	L	L	L	8
			H	L	L	H	9
			H	L	H	L	10
			H	L	H	H	11
			H	H	L	L	12
			H	H	L	H	13
			H	H	H	L	14
			H	H	H	H	15
High	X	Load Low	Follow inputs ABCD				0 → 15
Low	X	X	LLLL				0

As is:

Q_A — Div 2
 Q_B Q_C Q_D — Div 8

Fig. 9-2. Pin-out and state table for the LS197.

mode, the outputs follow the data lines transparently, because the preset and clear lines on each flip-flop are enabled to their respective data inputs. When clear is low the outputs remain low, regardless of the state of the other input lines.

The counterpart to the LS197 which counts in BCD or binary-coded-decimal format (0000 to 1001) is the LS196. The pin-out, which is the same as the LS197, and state table are shown in Fig. 9-3. An external jumper between pins 5 and 6 is likewise

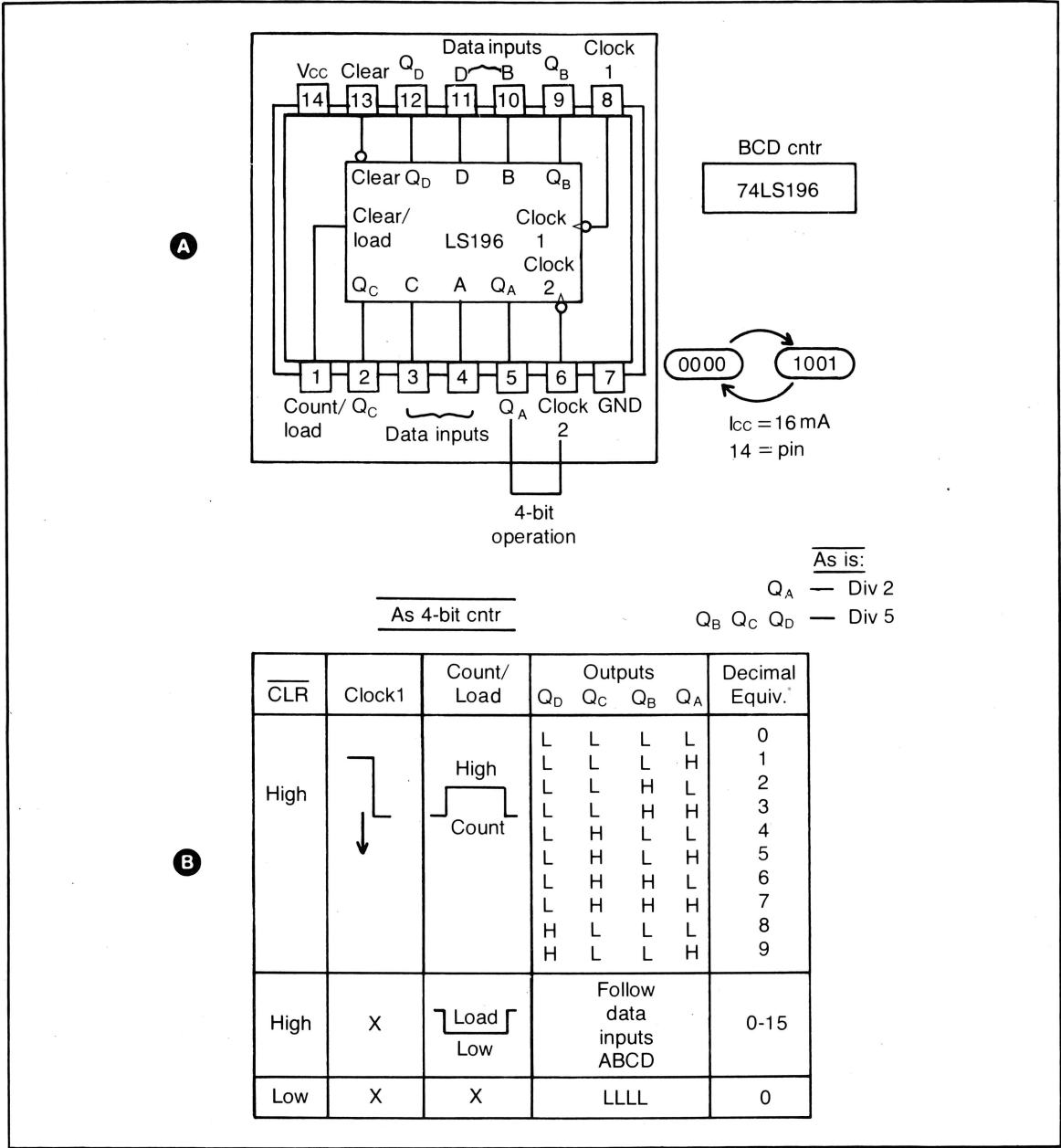


Fig. 9-3. Pin-out and state table for the LS196 BCD ripple counter.

necessary for 4-bit operation. As the state table shows, the count, load, and clear modes are the same. The only difference is that the device counts from decimal 0 to 9, rather than 0 to 15. Note that when acting as a transparent latch (load mode), the LS196 *will* output data greater than binary 1001/decimal 9.

Obviously, even these simple counters represent a considerable step up from the discrete SSI implementations of Experiment 14. With their small size and modest power requirements, and their ability to act as both counter and transparent latch, the LS196/197 would be much preferred over the SSI version.

However, while ripple counters are simple, they do have certain limitations. For instance, you'll note that there are limits on the operating frequency. With the external connection, the upper limit is something less than 15 MHz. Part of this limitation is due to the cumulative effect of propagation delay inherent in any ripple counter. Another consequence of the ripple counter is that the non-simultaneous transition of the outputs may, under certain critical conditions, cause problems in timing. A solution to such problems is provided by the design of *synchronous* counters, which you'll examine shortly. First, let's demonstrate the operation of the basic ripple counter.

EXPERIMENT 16, MSI RIPPLE COUNTER

Purpose

To show the operation of binary and BCD ripple counters, along with some gating circuitry.

Materials

- 1 - 74LS196 BCD counter
- 1 - 74LS197 Binary counter
- 1 - 74LS32 Quad OR

NOTE - Use STD-TTL if LSTTL is unavailable.

Procedure

1. The circuit of Fig. 9-4 will serve as a guide for both the BCD and binary four-bit ripple counters. The utility strobe is used directly as a clock, though

you could interpose a toggle flip-flop with an LED as done earlier. In this configuration, all the annunciator lines are employed as data inputs, to demonstrate the transparent latch or loading mode. This necessitates the use of jumpers to load and clear the device, as indicated.

2. At this stage you should be able to set up labeled headings on the BDIS display, if you wish. Confirm the various modes (clear, load/latch and count) of both devices. Try the LS197 binary counter first. Then, turn off the computer, replace the LS197 with the LS196 BCD counter, and proceed as before. What happens when you load the LS196 with a number greater than 1001, and then start counting up from there?

3. Problem: Can you think of a way to add an enable line? More specifically, gate the clock input, so that it can be enabled or disabled. Alternatively, you might want to isolate the clock signal in some way. Think about it. Two possible solutions are shown in Fig. 9-5.

Discussion

The operation of these ripple counters is straightforward, and serves as a basis for understanding the synchronous counters you will look at next. The little embellishment suggested in step 3 was intended to review your knowledge of earlier material. More challenging problems will be presented in later experiments.

SYNCHRONOUS COUNTERS

A universal four-bit counter is the synchronous up/down counter illustrated in Fig. 9-6 through 9-8. It comes in both binary and BCD varieties (LS193 and LS192). The pin-outs of each are the same and are given in Fig. 9-6A. This more useful type of counter can be understood as an extended version of the ripple counter just covered.

Like the ripple counter, the synchronous up/down counter can operate in three major modes: count, load, and clear. The important differences are as follows:

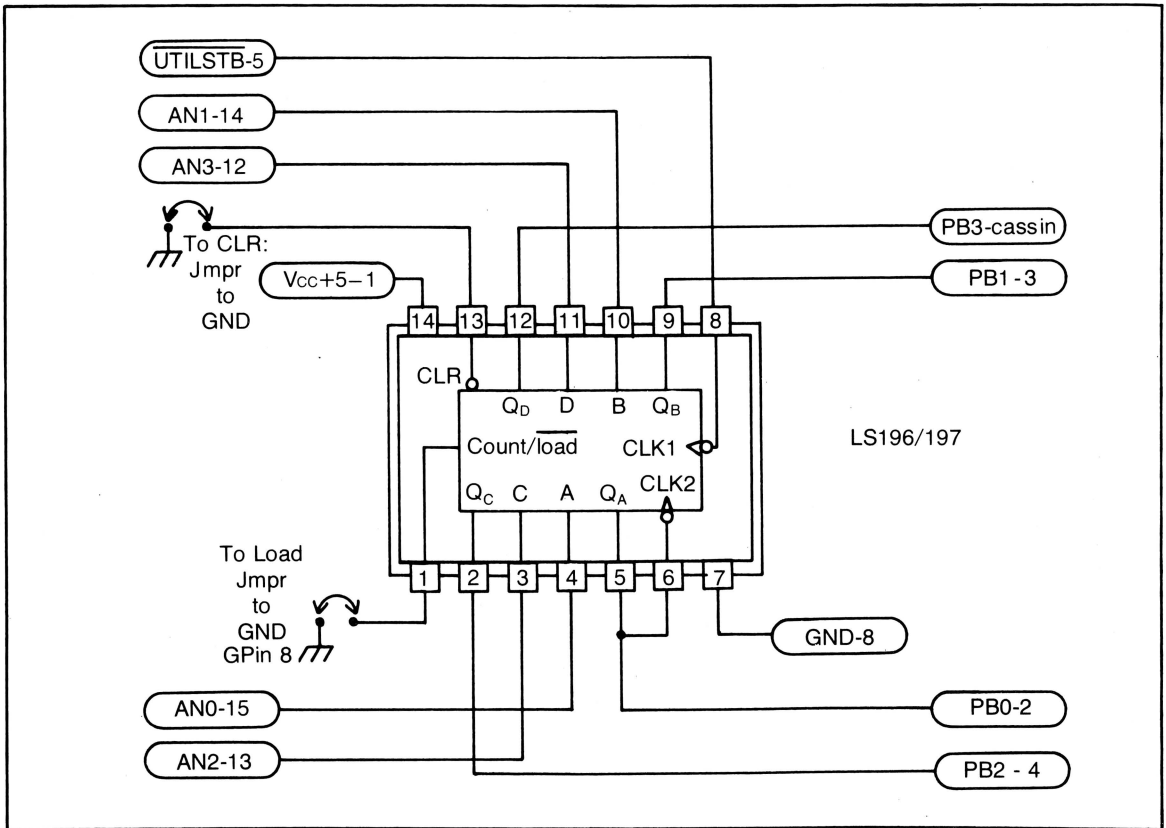


Fig. 9-4. Experiment 16 setup for LS196/197 demonstration.

□ It can count up or down, that is, from 0000 to 1111 or from 1111 to 0000 (taking the LS193 binary counter as an example). This is done simply by connecting the clock source to either the up or down inputs, pins 5 and 4, respectively.

□ The device counts synchronously. The flip-flops change state on the up-going edge of the clock. They are positively edge-triggered, like the flip-flops in the LS196/197. But beyond this, all flip-flops change simultaneously. Synchronous operation allows for more reliable operation in critical timing situations and for an operating frequency of 32 Megahertz. That is more than twice the frequency of the ripple counter!

□ There are additional outputs—carry and borrow—which permit cascading of the counter for multidigit operation. This is necessary in such applications as digital clock displays and in test

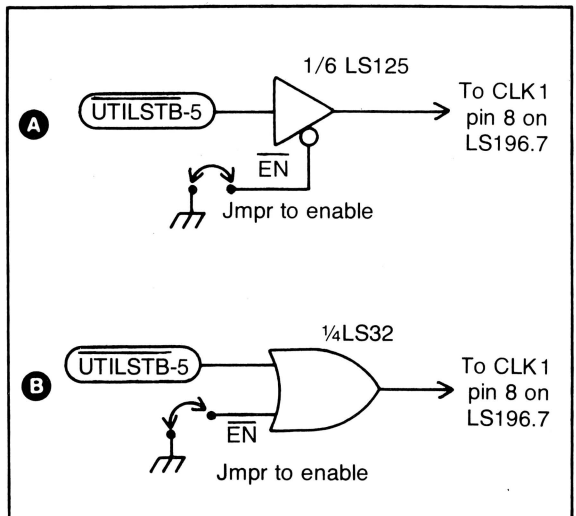


Fig. 9-5. Two methods of enabling/disabling the clock signal to the LS196/197.

equipment such as DMMs. Also, the borrow output can be employed for a divide-by-N (also called modulo-N) mode of operation, which will be explained later.

Clear and load input lines operate as they do in the ripple counter, either setting the outputs to zero, or allowing the device to operate as a latch.

Remarkably, all of these features are packed into a 16-pin DIP, which consumes only 19 mA of supply current when the outputs are unloaded. This is only slightly more than the ripple counter! And the price, currently, is only about fifty cents via mail-order in single quantities. This powerful little four-bit universal counter is a prime example of the appeal of MSI designs. (Just imagine constructing the circuit of Fig. 9-6B from individual SSI components).

More detail on the function of the binary synchronous counter is provided in Fig. 9-7. The timing diagram is, in this case, a much more convenient and meaningful way of illustrating the modes of operation than a state table. The diagram depicts a sequence of operations:

1. Clear outputs to 0,
2. Load with decimal 13 / binary 1101,
3. Count up as follows: 14 - 15 - 0 - 1 - 2,
4. Count down as follows: 1 - 0 - 15 - 14 - 13.

Referring to the timing diagram, you'll note that the clear line is active high. It is an asynchronous input and overrides all other inputs. Next, the counter is preset to 1101 by the load line, which is an active low input. While load is low, the outputs will follow the inputs transparently. Then, the clock signal is fed into the count up input. When the counter reaches 15 (binary 1111), the carry output goes low for half a clock cycle. The count continues to decimal 2. Then the clock signal is fed into the count down input. When the counter reaches zero (binary 0000), the borrow output goes low for half a clock cycle, and the count finishes where we began (binary 1101).

A few comments regarding the details of the transitions and about the borrow and carry lines are

in order. First, the output state of the counter changes on the leading or up-going edge of the clock. This is because the individual flip-flops are D-type devices, not master/slave flip-flops. Second, the borrow and carry outputs go low on the trailing or down-going edge of the clock and remain low until the clock signal again goes high. The low pulse on carry occurs after the output has achieved 1111 for a count up operation. The low pulse on borrow occurs after the output has achieved 0000 for a count down operation. These aspects of carry and borrow transition are illustrated in the bottom of Fig. 9-7.

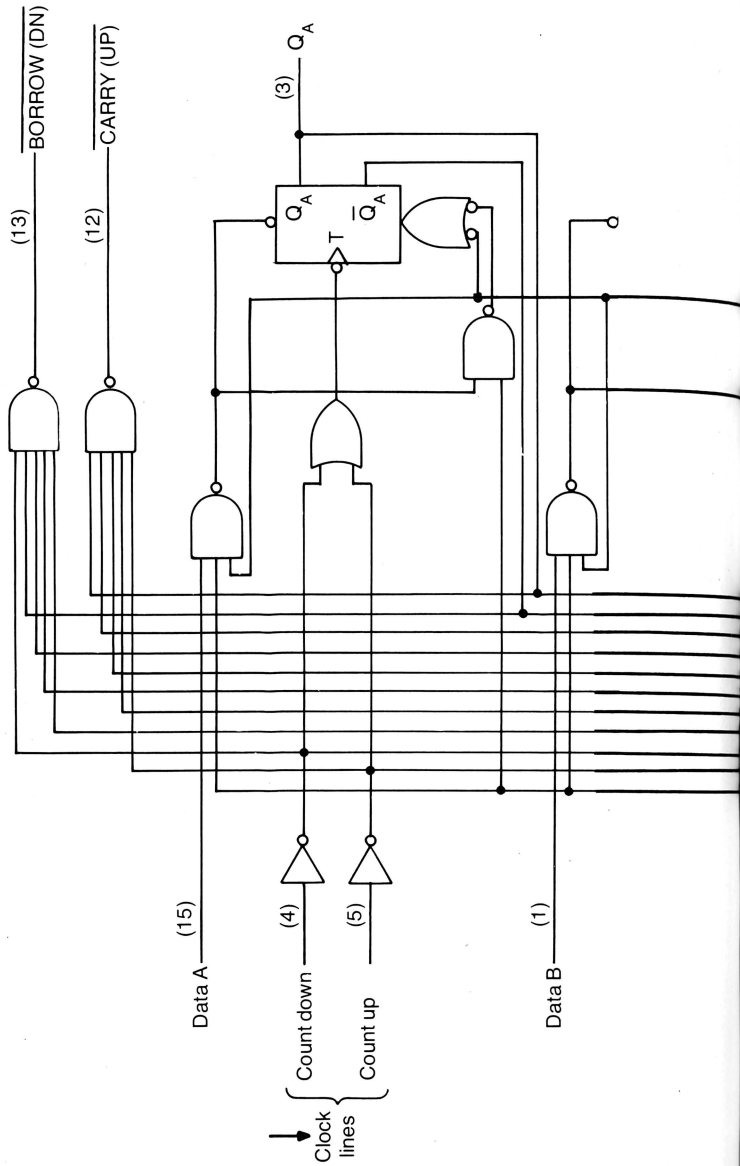
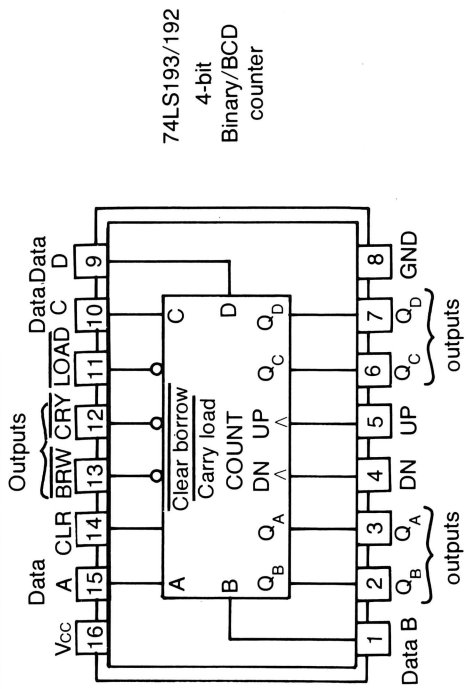
As mentioned, the synchronous up/down counter comes in a BCD version, the LS192. The timing diagram for this device is given in Fig. 9-8. As with the diagram for the binary counter, this diagram illustrates the major modes of operation: clear, load, count up (with carry), and count down (with borrow). Examine this figure as you did the previous one. Note that the borrow and carry outputs are again active low when the counter has counted down to 0000 or up to 1001.

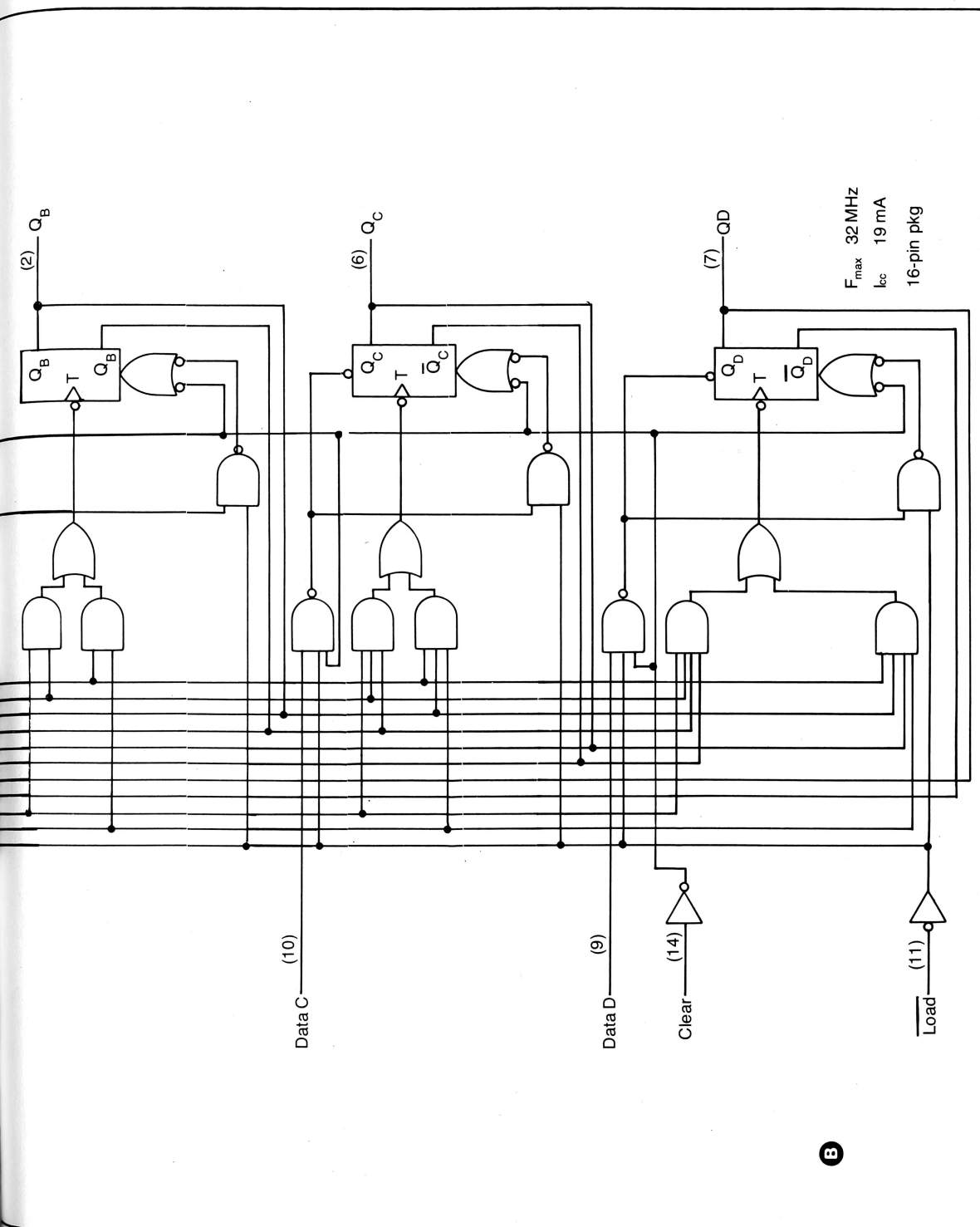
Two very simple applications will be mentioned: modulo-N division, and multidigit counting.

One use of the synchronous up/down counter is as a *modulo-N*, or divide-by-N device, where N is any number between 2 and 30 in increments of 2. If you connect the borrow output to the load pin, and feed the clock signal to the count down clock input, as in Fig. 9-9A, you have a modulo-N configuration. The operation is as follows: Assume the output state is 0000, and the clock goes low. Then borrow goes low, and so therefore does the signal on load pin, to which it is connected. Then, whatever data is on the inputs is loaded into the counter. When the clock resumes with the next low to high transition, counting down from this preset number will occur. When the counter reaches 0000, that number will again be loaded (if it has not been changed in the meantime) and the cycle repeats. At any time you can change the data inputs in order to obtain division by a different number.

Note that the division is taken with respect to the frequency of the clock pulse, so that the minimum divisor is 2, and the maximum is 30. The

A





B

Fig. 9-6. Synchronous counters: pin-out and schematic diagram of the LS192/193 (BCD and binary) synchronous counters. These are universal MSI counters which are powerful for their size, and also in regard to both package cost and power consumption.

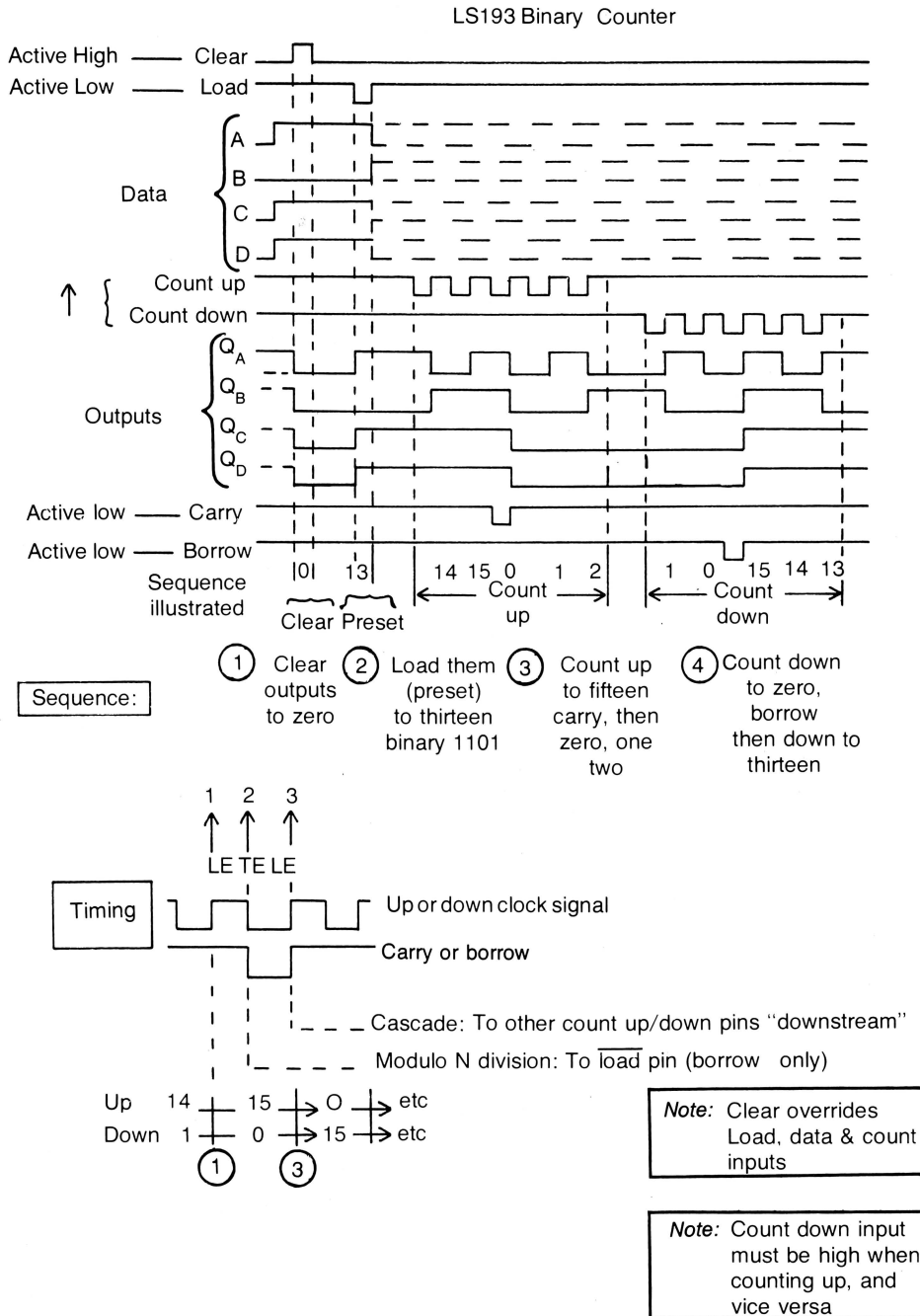


Fig. 9-7. Detailed timing diagram explains the operation of the LS193. See text.

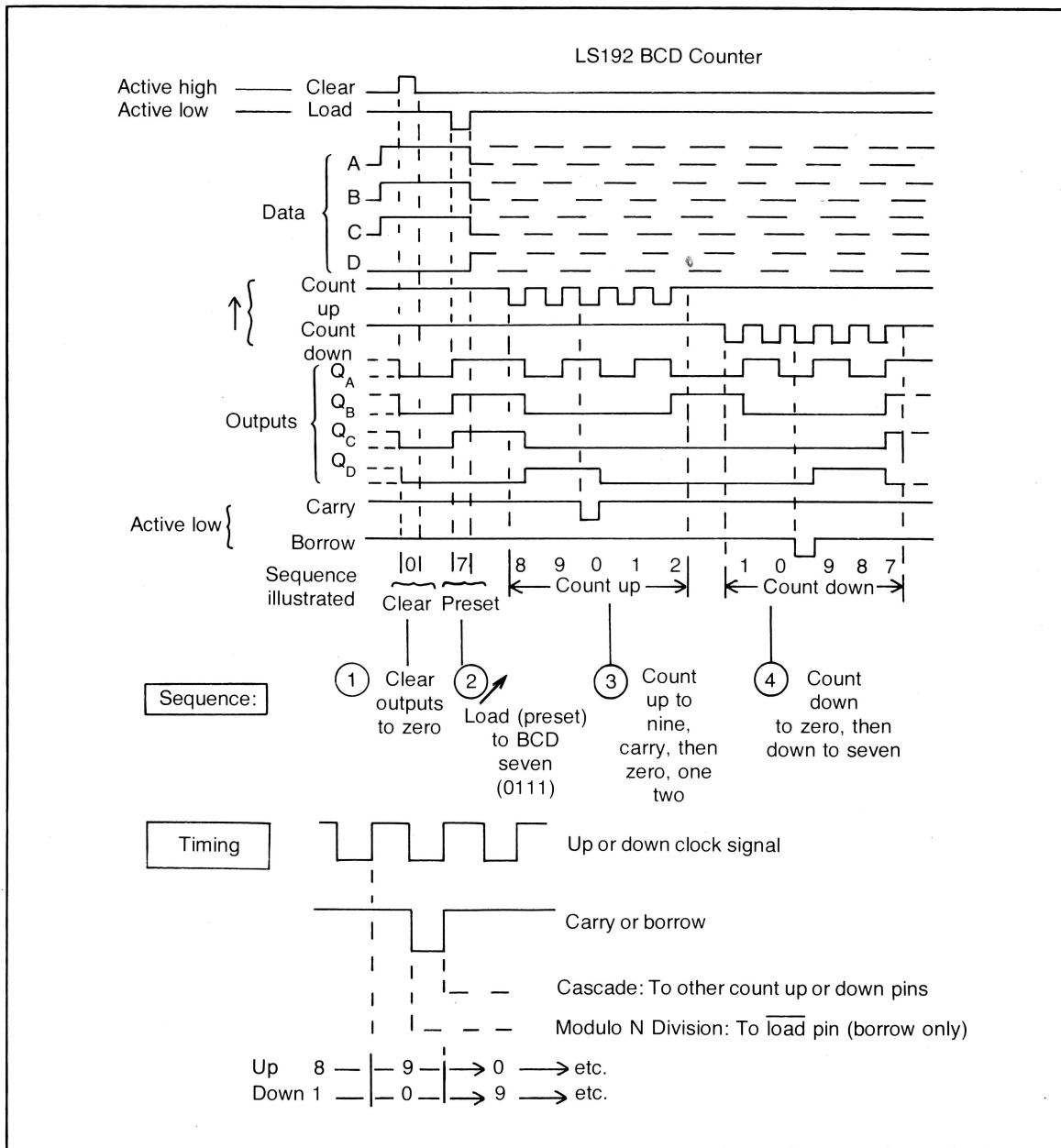


Fig. 9-8. Detailed timing diagram for the LS192 BCD device. It is similar to the LS193 except for the counting range.

actual output is taken off the borrow pin itself. It can be inverted if need be.

A second application is that of cascading the counters for multidigit counting. For instance, two LS193s may be connected as shown in Fig. 9-9B.

The clock signal is fed into the count up pin on counter A. The carry output off counter A is connected to the count up pin on counter B. Each time the least significant four bits of counter A have achieved binary 1111 and begin the next cycle, the

carry line of A goes low then high, and counter B will increment by one. This is a cascade configuration, and in fact, the count in device B will increment half a clock cycle later than the count in device A. Therefore, this two digit counter is not truly synchronous.

For two LS193 devices, the count can range from 0 to 255 (or hex \$00 to \$FF). Two LS192 devices could count from decimal 0 to 99.

Finally, there are other options to the LS192/193. The LS190 BCD and LS191 binary counters are both four-bit synchronous up/down counters.

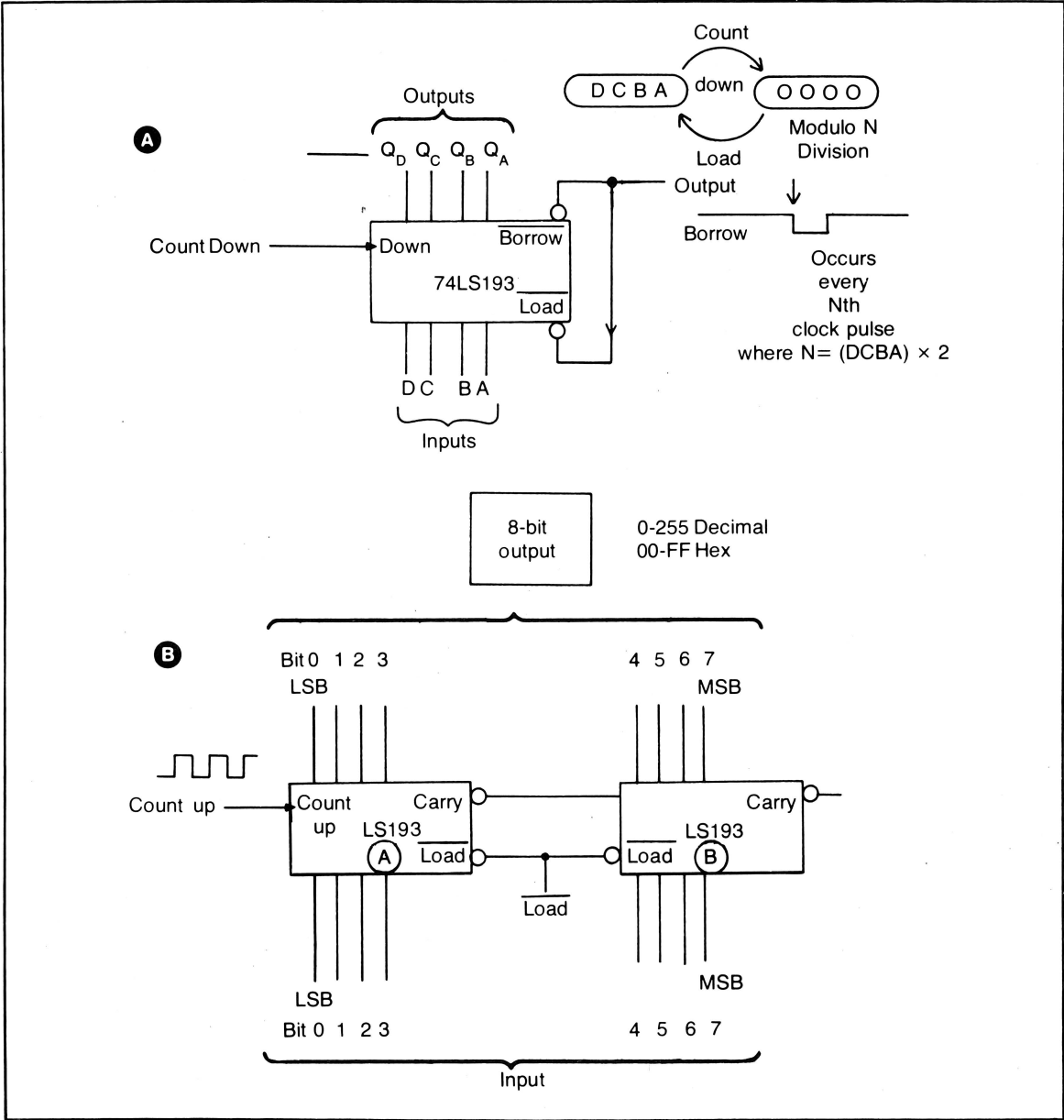


Fig. 9-9. Applications of the LS193 as (A) a modulo-N device, and (B) in multidigit counting.

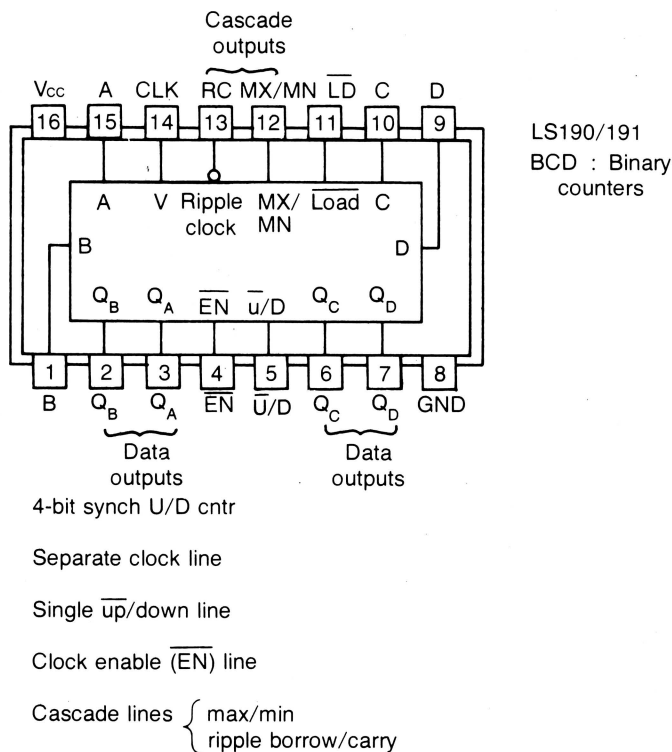


Fig. 9-10. Another four-bit synchronous up/down counter. BCD and binary versions (LS190 and 191) have that same pin-out. The somewhat different features from the LS192/193 are listed.

However, they differ from the counters we just discussed in a few respects. The major difference is that these devices have an up/down control pin, and a separate clock pin. This means that the device can be made to change the direction of its counting instantly, with no necessity of reconnecting the clock signal from a count up to a count down pin, or *vice-versa*. This is a distinct advantage in some control applications, where reversibility is required. To count up, a low must be placed on the up/down input.

In addition, there is a separate enable pin, which is active low. This allows counting to be disabled or enabled at will. Last, there are cascade lines—max/min and ripple borrow/carry. These are used for modulo-N functions, as well as for cascading for multidigit counting and display applications.

In some respects, the LS190/191 may be a more flexible device, since the only functions lacking is an asynchronous clear line. However, there are problems if you try to disable the counter, or change the up/down mode, while the clock is low. The exact nature and reason for such problems will be explored in Experiment 17. You should refer to data and application manuals for further information on this counter.

EXPERIMENT 17, THE SYNCHRONOUS UP/DOWN COUNTER

Purpose

To show the basic operation of a synchronous up/down counter. A short project involving the addition of control gating circuitry is included. Limitations of synchronous counters are discussed.

Materials

- 1 - 74LS193 synchronous up/down binary counter
- 2 - 74LS192 synchronous up/down BCD counter (optional)
- 1 - 74LS74 D-flip-flop
- 1 - 74LS00 quad NAND
- 1 - 74LS32 quad OR
- 3 - 330 ohm resistors

NOTE - Use STD-TTL if LS192/193 unavailable.

Procedure

1. The circuit shown in Fig. 9-11 will allow you to demonstrate all of the functions of the LS193 4-bit binary counter and the LS192 BCD counter. The utility strobe is used as the clock. It may be attached directly to either the up or down clock pins

to demonstrate the bidirectional counting capability. However, the edge-triggered LS74 flip-flop with LED indicator is included in this circuit so that you can see how the output state changes on the up-going clock transistion. (This corresponds to the LED turning on). The two other LEDs indicate the status of the borrow and carry pins during counting. Jumpers for load and clear are also indicated.

2. As you recall, the basic functions of the synchronous up/down counter are: clear, load, count up, and count down. You should leave clear grounded while counting and loading; tie it high to set the outputs to zero. You should be able to leave load floating high/inactive during counting; however, tie it high to +5 V if you experience any problems with erratic operation. As you demonstrate these various modes, you might want to

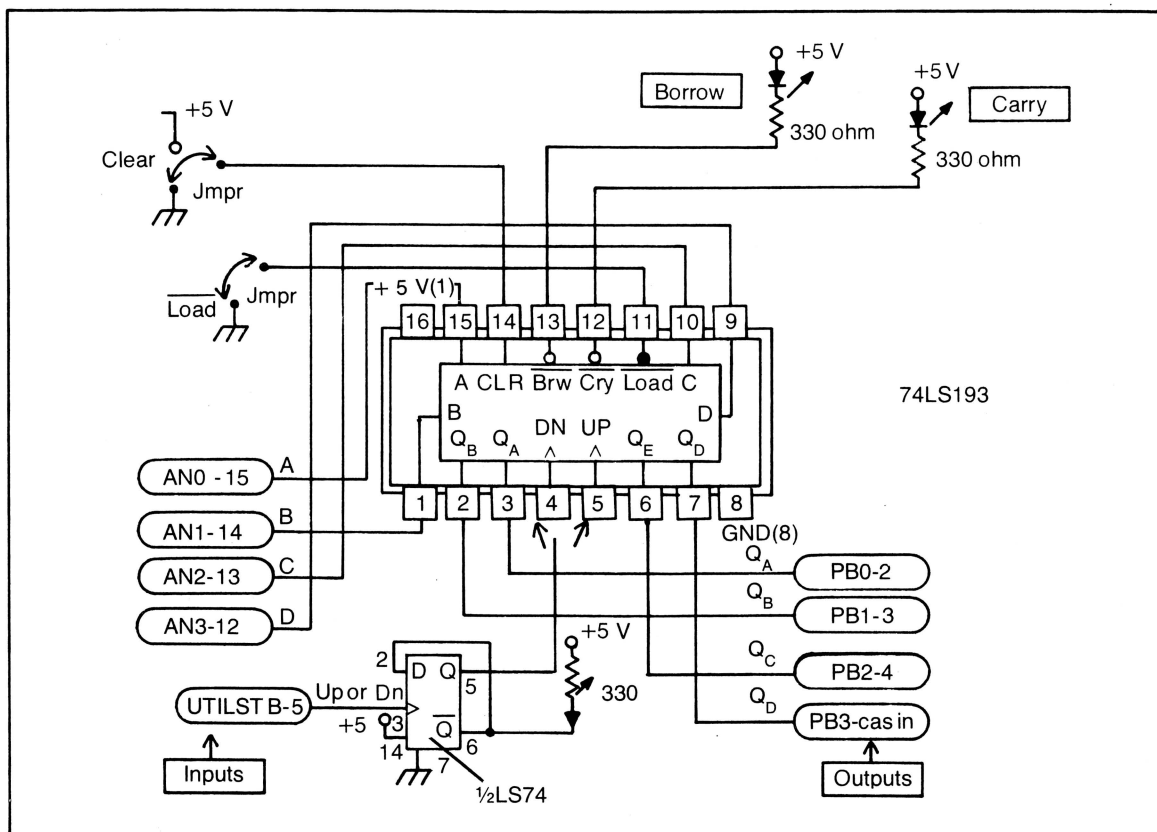


Fig. 9-11. Experiment 17 setup for the LS193 demonstration.

follow the example of the timing diagram given earlier. After observing the binary LS193 device, you can substitute the LS192 binary counter. What happens when you load a number greater than 1001, and then start counting?

3. You may want to show the modulo-N or divide-by-N function of this counter by connecting the borrow pin to the load pin. Follow the schematic in Fig. 9-9A. When, during the clock cycle, does the loading action occur?

4. Control circuitry project: Let's pose a simple design problem. Suppose you wanted to add the following features to the LS193 counter: (1) An enable line, which would allow you to disable the

counter so that clock pulses would have no effect on the count state. (2) A single up/down line, which when high would effect a count up operation, and when low would cause the counter to count down. A separate clock line would be needed in this case. Consider how you would approach this problem before you proceed.

5. Defining the problem: As you think about this little project, you may find it convenient to draw a block diagram that includes the inputs and outputs of the circuit you want to design. This is indicated by the block diagram in Fig. 9-12A. The up/down control line, which we'll call count up (CUP), the enable line (EN), and the clock line (CK) are shown. Since the LS193 and 192 have separate up and down pins, two outputs from the control circuitry are

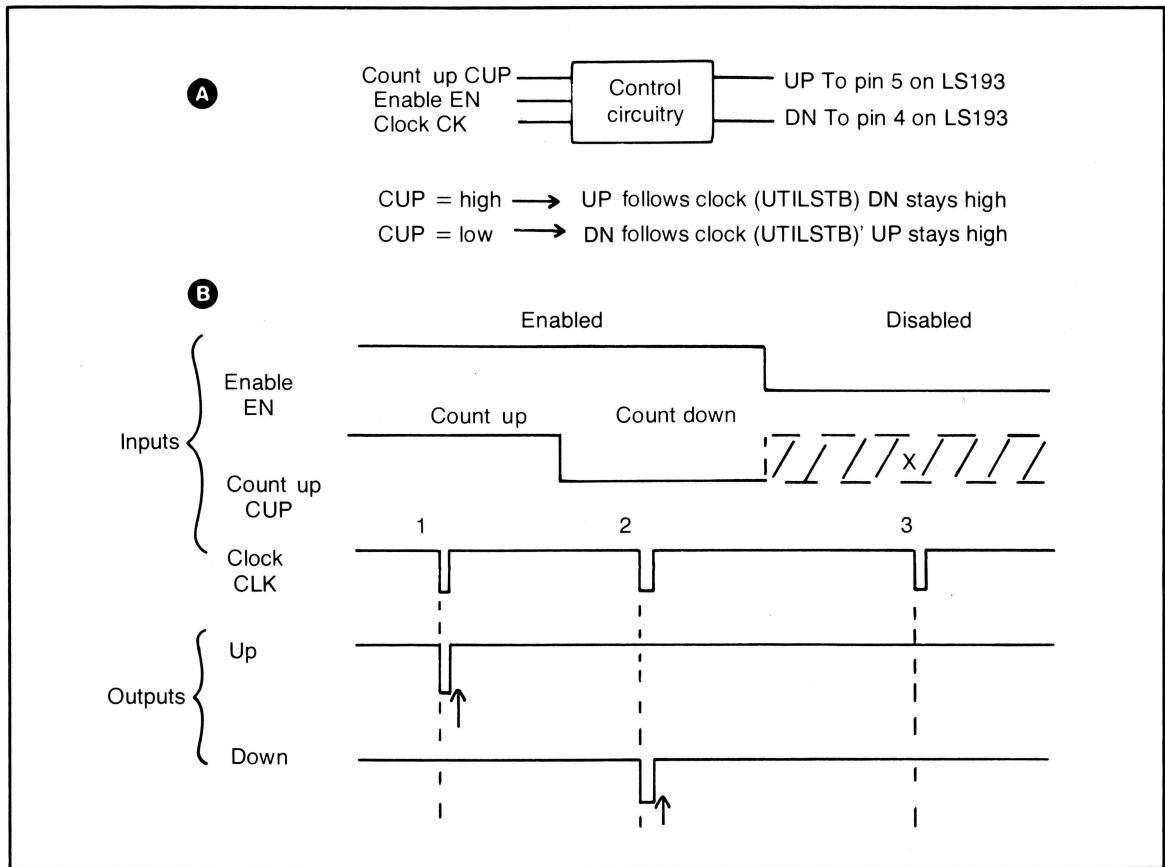


Fig. 9-12. Statement of the control circuitry design problem of Experiment 17 given in block diagram and timing diagram form.

A

		EN	CUP	CLK	UP	DN	
Count up	m0	H	H	H	H	H	$EN \cdot \overline{CLK} \cdot CUP = \overline{UP}$
	m1	H	H	L	L	H	
Count down	m2	H	L	H	H	H	$EN \cdot CLK \cdot \overline{CUP} = \overline{DN}$
	m3	H	L	L	H	L	
Disabled	m4	L	X	X	H	H	

B

$$\overline{(EN \cdot \overline{CLK}) \cdot CUP} = UP \quad \overline{(EN \cdot \overline{CLK}) \cdot CUP} = DN$$

$$EN \cdot \overline{CLK} + CUP = UP \quad EN \cdot \overline{CLK} + CUP = DN$$

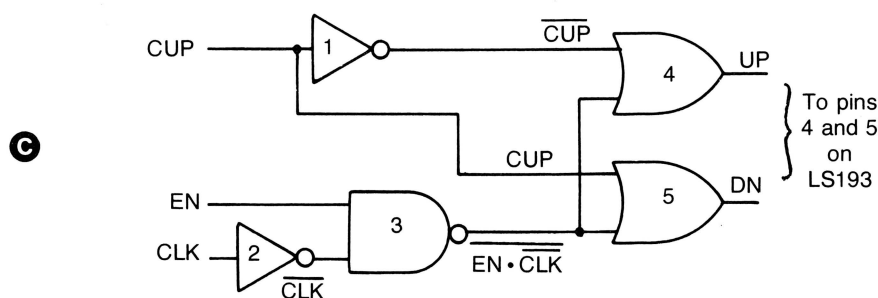


Fig. 9-13. Solution to design problem: truth table with two equations (for the two outputs from the control circuitry), along with the corresponding logic schematic.

necessary. Naturally, if we select a count up mode, we would like the up output to follow the clock, while the down output should not change. In such a case the down output would remain at the same logic level, let's say high. Similar reasoning applies when we want to count down.

6. A timing diagram often helps, even in such a simple application as this. There are three inputs and two outputs to and from the control circuitry, so a five-line timing diagram should allow you to specify all the states desired. Try constructing one, and then refer to Fig. 9-12B.

7. A truth table is the next step in the design

process. You should try to translate the timing diagram directly into such a table. Refer to Fig. 9-13A. The sense of the table can be summarized briefly: When both EN and CUP are high, the up output should follow the clock input, and down should remain high. Similarly, when CUP is low, the down output should follow the clock, and up should remain high. When EN is low, both up and down should not change with the clock. They should stay high.

8. Now you can write the Boolean equations using the appropriate minterms from the truth table. You could use the output high states to write the equations. However, with only one low state for each

output, it is more economical and faster to use these states for the final equations. Minterms m_1 and m_3 are used. Note that the input terms are equated to the negated or inverted output. Figure 9-13B shows the rearranged equations. De Morgan's Theorem is invoked.

9. The preliminary circuit is given in Fig. 9-13C. This is really a direct translation from the equations. You can reduce package count by using NAND gates in place of the inverters.

10. The final circuit and pin-out for this problem is given in Fig. 9-14. Demonstrate the operation of the control circuitry by first setting the enable line high, and then alternatively counting up and down at will. Clear and load operate as before, however, binary 1111 will be loaded when pin 11 is grounded if you leave the data inputs open/high as indicated.

11. There are two bugs in this particular design. Try enabling and disabling the counter while in up and down count modes. Specifically, stop the count alternatively when the clock is high and low, then disable and enable the device. Change the count mode as an alternative (up and down and vice versa). Do this several times.

Do you notice a problem when you disable the device by setting EN to low, or when you change the up/down count mode. What happens? Is there a quick solution to these problems?

Discussion

In the first portion of this experiment you looked at the basic modes of operation of the synchronous up/down counter. When using the BCD LS192 device, you may have tried to load a number greater than 1001. While this is illegal as far as the BCD format is concerned, the outputs do register such data. When the counting is continued, the device proceeds (either up or down, as the case may be) until it is in the normal BCD range of 0000 to 1001. From there on, it stays in this range.

Hopefully, you tried the modulo-N circuit, as suggested. The intention here was simply to re-

state, by demonstration, some of the aspects of device timing. In particular, you may have noted that the borrow line went low when the count reached zero and the clock went low (trailing edge). First, the counter reached 0000 on the leading edge of the clock (rising edge), and borrow was still high/inactive. Then, on the falling edge of the clock, borrow went low (the LED lit up). That is, as you toggled the clock line, the active low of borrow occurred $\frac{1}{2}$ clock cycle after the 0000 state. Only then was the number on the data lines loaded into the counter. This $\frac{1}{2}$ clock cycle between 0000 and borrow going low delay was intentionally designed into the counter because without it the 0000 state would exist only for the duration of the propagation delay through the counter, a few tens of nanoseconds at most.

Finally, in the last part of the experiment, a design problem was stated. One purpose was to provide a quick review of the steps in SSI combinational design discussed a few chapters back.

Another purpose was to illustrate the little surprises that crop up in what seemed otherwise logical design solutions. There were hidden bugs in the resulting circuit which you undoubtedly encountered. Specifically, if you disabled the counter by bringing the EN line low (inactive) while the clock line was in the low state, the counter either advanced or decremented one count, depending on whether you had been counting up or down at the time. Also, if you shifted the CUP (up/down control) while the clock line was low, a false count may have occurred. Examining the truth table shows why. What, if any, are the solutions to these problems?

Taking the second problem first, the solution is trivial: do not shift the count up/down control line (CUP) while the clock is low. There is really no easy way around this problem, because it is an inherent limitation in this type of control circuitry.

Now let's tackle the problem of false counting that occurs when the device is disabled with a low on en. One might be tempted to write out the truth table in full for the disabled state (EN low) specifying that no such transition can occur in the respective up and down outputs. Ideally, you would want

the up and down lines to remain in the state they were originally in before being disabled. This would require two more equations, and more complex control circuitry.

Another solution would be to enable and disable the control gating circuitry with three-state devices connected to the up and down lines with the signal EN fed into the three-state inputs. Again, more circuitry.

A much simpler solution is to utilize an extremely short negative-going pulse as the clock signal. This signal would be normally high, but when active would go low for a fraction of a microsecond. In fact, you have such a clock signal available: simply use the utility strobe directly off of game port pin 5! Use it in place of the AN0 clock source in Fig. 9-14. Unless you disable the device during the $\frac{1}{2}$ microsecond low interval of the utility strobe pulse, there will be no problem with false counting. While this is not a 100% solution, the use of a very narrow clock pulse does lessen the danger of false counts when disabling the counter in an actual application.

Note that the conflict of inputs illustrated above is a problem inherent in many sequential devices, not just our simple control circuit. If you read the data on the previously mentioned LS190/191 counter, which has integral up/down and enable pins as part of the device, you will see cautions stated regarding false counting during the time when the clock line is low. By virtue of this exercise, you now have a real understanding of this type of limitation of MSI counters and the admonition given in the data manuals (TI's in this case).

SHIFT REGISTERS

Related to the MSI counter is the shift register. Like the counter, a typical MSI shift register is four bits wide. Larger ones fall into the LSI range. As the name suggests, the shift register is not a static register which simply holds and stores data like a latch, but rather it is a dynamic device which can shift data bits between the flip-flops which constitute it.

Basic Operation

A four bit shift register is composed of four D-type flip-flops and associated gating circuitry for a total of 40 or 50 gate equivalents. This places it in the middle of the MSI level of complexity. Each flip-flop can be considered as a temporary storage bin which can contain one bit of data. Note that the bins have been labeled A through D with A the most significant bit and D the least significant bit. An array of four such bit bins will have a serial data input (SI), a clock (CK) line to serially load or shift the data in one bit at a time, and a serial data output (SO).

The basic action is illustrated in Fig. 9-15A through E. The register shown is a serial-in/serial-out device; for simplicity, parallel input and output lines have been omitted.

In Fig. 9-15A we begin with the register in the 0000 state. A logic high/1 is present on the serial-in data line, SI. Along comes clock pulse 1. The binary 1 is loaded on the leading edge of this pulse, and the result is given in Fig. 9-15B: a binary 0 appears on the serial output, SO. We then place a 0 on the SI line, deliver the 2nd pulse on the clock line, and the result is as given in Fig. 9-15C. Two more clock pulses, with 0 and 1 data inputs, respectively result in the final state shown in Fig. 9-15E.

What we've done is serially load a binary 1001 (decimal 9) into the register, and at the same time shift out a binary 0000. This operation is also called a *shift right operation*, with the direction from A to D (most to least significant bit bin).

Shift register operation is, then, pretty straightforward. Now let's add a few embellishments.

Figure 9-16 illustrates another four-bit shift register. This device, however, is a full blown universal register. We'll use the 74LS194A as it contains most of the features of a typical universal shift register. With it, by setting mode controls, you can do the following:

- Serially load data from the left, shifting data to the right, as just done in the prior example. The data input is referred to as the serial shift right input

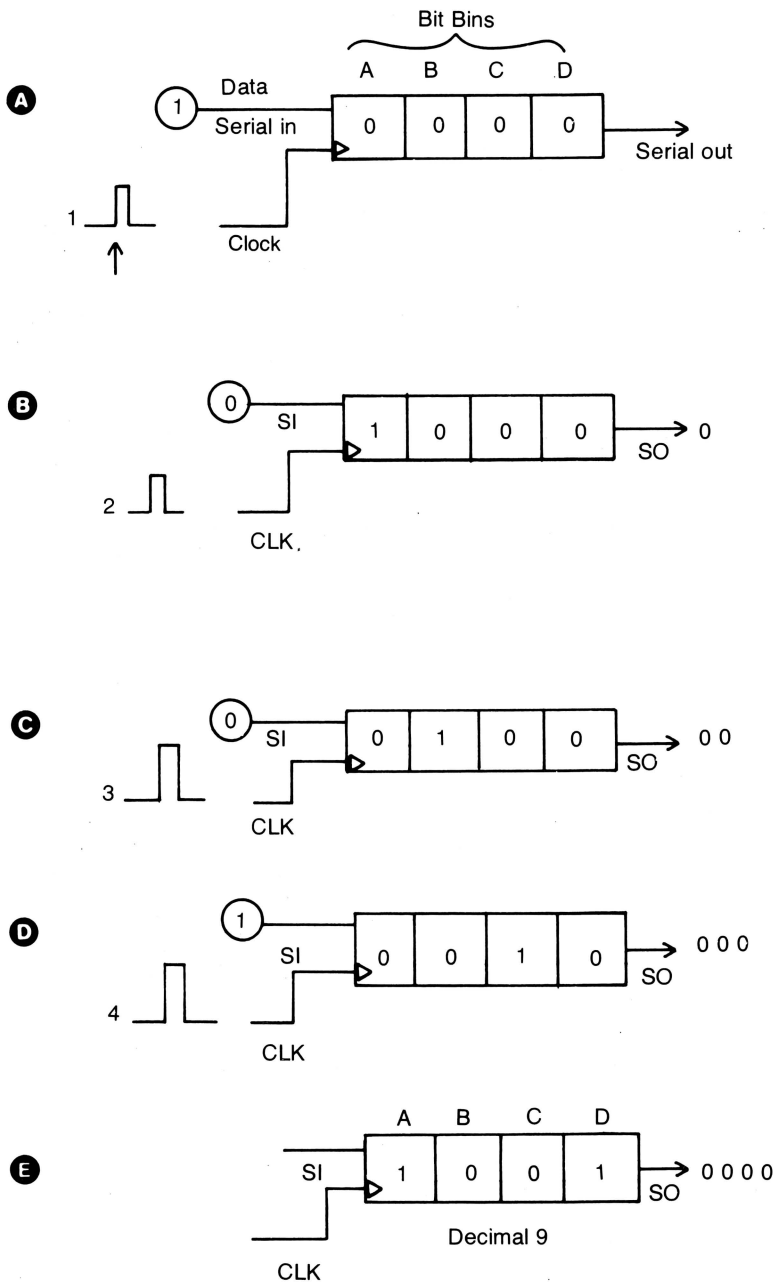


Fig. 9-15. Simplified representation of shift register operation, shift right in this case.

or SRT for short. Take particular note that there is no serial output line labeled as such. Rather, since data is being shifted in the direction from Q_A to Q_D , we use output Q_D as the serial shift right output or SRO.

□ Serially load data from the right, shifting data to the left. In this shift left mode, the data line used is called the serial shift left input, or SLT. Since data shift from right to left— Q_D to Q_A —serial output data is taken from Q_A , the serial shift left output, or SLO.

□ At any time during or in between the shift left or shift right operations, data can be read in parallel from the data outputs, Q_A through Q_D , as a four bit number. This requires no special mode control.

□ By selecting the proper mode, you can also load data in parallel, that is, as a four bit number via the data input lines, D_A through D_D .

□ You also have the option to turn off or disable the device so that no parallel loading or shifting in either direction can occur.

Mode controls were mentioned above. These are the two lines labeled S_0 and S_1 in Fig. 9-16A. They allow you to select any one of four modes: inhibit, shift right, shift left, and parallel load. The table for these four modes is given in Fig. 9-16B.

The pin-out of the LS194A universal four-bit shift register is shown in Fig. 9-17. You'll note that it contains one more function we did not mention in the above, namely, an active low clear. This is provided on pin 1.

As to the power requirements of this device, they are really quite modest. The nominal value of I_{CC} is 15 mA. Again, we have another impressive array of features in a small package that is typical of the MSI group.

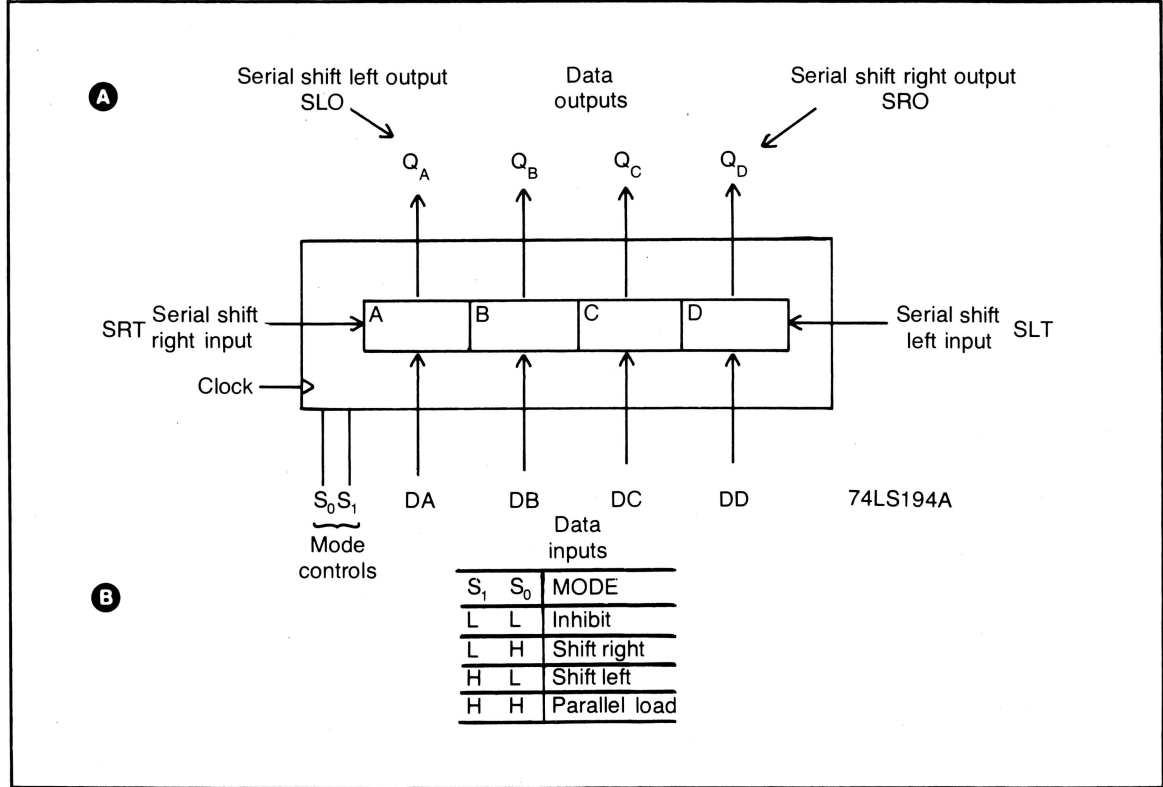
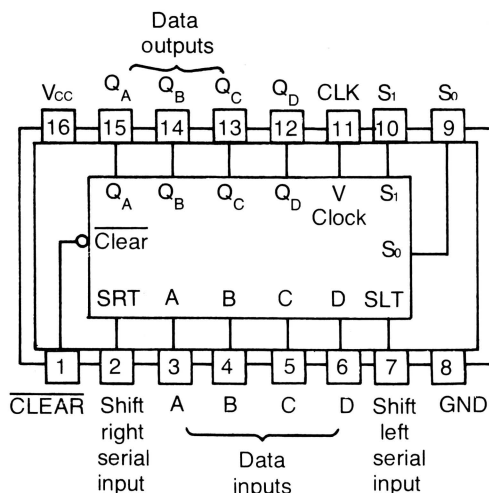


Fig. 9-16. More elaborate LS194A universal shift register with bidirectional operation and parallel inputs and outputs. Mode control states are given in (B).



74LS194A

$I_{CC} = 15 \text{ mA}$
(nominal)

Fig. 9-17. Actual pin-out of the LS194A.

Shift Register Applications

There are several applications for shift registers which we'll mention; some of which may already be obvious to you. They are:

- ☐ Serial-to-parallel conversion.
- ☐ Parallel-to-serial conversion.
- ☐ Multiply and divide by two.
- ☐ Sequence generation.

Serial/Parallel Format Interconversion.

Data in digital systems is usually transmitted in either of two formats: serial or parallel. Shift registers are key elements in the circuit that provide interconversion between these two formats.

Parallel data is usually organized into words of various sizes from the 4-bit nibble to the 8-bit byte, and larger. System buses carry data in the parallel format. The advantages of parallel data are its compactness and economy. The main disadvantage is that parallel lines can transmit data over a rather limited distance. Parallel "rainbow" or plain flat ribbon cable has a limit of little more than a yard. The problem stems from transmission line effects, such as reflections up and down the line, and particularly crosstalk between lines in the same cable.

(Remember capacitive shunting effects between parallel conductors?)

Serial data, on the other hand, can be transmitted over much longer distances, as for example, through the so-called *twisted pairs* consisting of a signal line and a ground line. Digital data transfer over telephone lines is a prime example of serial data transmission. The problem here is that serial data schemes are complex and costly because they entail a fair amount of circuitry to convert parallel data into serial format on the sending end, and then back again into parallel format on the receiving end. Serially transmitted data is also inherently a lot slower, however this may not be a serious drawback in certain situations.

Each format has its proper place in the scheme of things. In a self-contained computing system where speed, economy and small size are essential, the parallel bus organization is the rule. When data generated in that system must be transferred over some distance, parallel-to-serial (P-to-S) and serial-to-parallel (S-to-P) conversion circuitry is used because of the inherent long distance advantage of serial data.

Obviously, shift registers of one sort or another play a key role in the transmission of digital

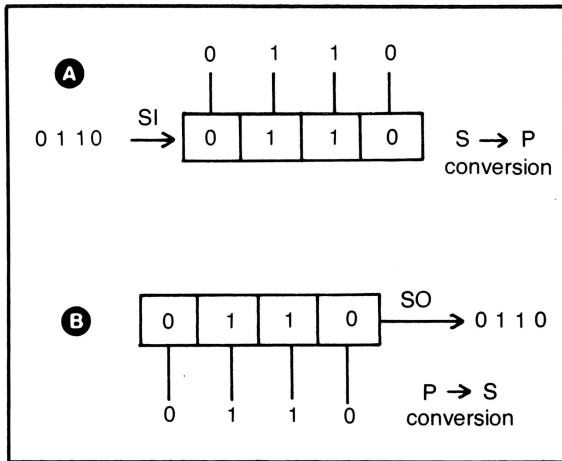


Fig. 9-18. Application of shift register in S-to-P and P-to-S conversion.

information. As Fig. 9-18 illustrates, the universal shift register can perform both S-to-P and P-to-S conversion. Note that a bidirectional shift register (shift right and left modes) can act as either a send or receive device from or to the digital system. In the figure, the conversion of binary 0110 (decimal 6) is illustrated. The principle can be readily extended to eight and sixteen bit words.

Multiply and Divide by Two. An important function in both microprocessor chips and special, dedicated arithmetic processor chips is circuitry to perform binary multiplication and division. The simplest operation is a multiply or divide by two. This is accomplished easily enough by a shift left or shift right operation, as given in Fig. 9-19. Obviously, the number shifted cannot be allowed to simply fall out: it must be stored as either the most or least significant result of the multiply or divide operation. Therefore, additional shift register elements and control circuitry are required in the typical arithmetic circuit.

The subject of arithmetic processing in hardware using ALUs and other, more advanced chips (so-called coprocessors) would fill several chapters if not a book of advanced material. But at least you can see how shift registers play a role in such devices.

Sequence Generators. This application is depicted in Fig. 9-20. Here, the contents of the

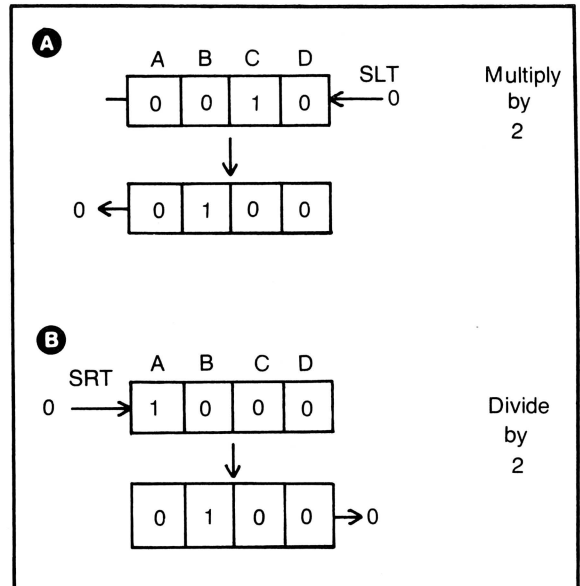


Fig. 9-19. Application in multiplication and division by two.

register are *recirculated* from the serial output back into the serial input. Each time the clock signal occurs, a bit is shifted out of the output and back into the serial data input. The train of data coming out can be read as the final output, and this would consist of a recurring pattern of bits, that is, a sequence of high and low logic levels. You might also think of the device as a wave form generator, with a square wave output as shown.

Naturally, longer *bit trains* can be manipulated

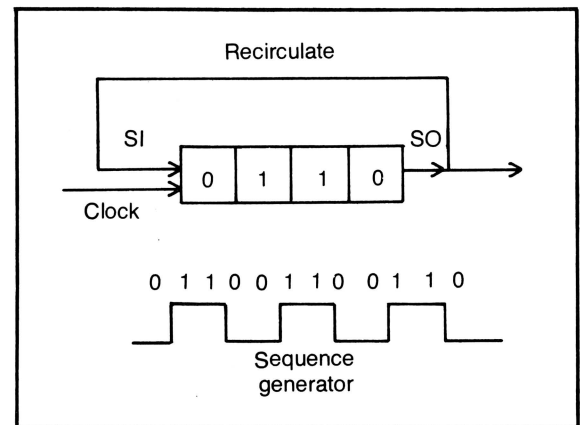


Fig. 9-20. Applications as write/recirculate device, or sequence generator.

by using larger shift registers or by stringing several smaller registers in series.

Modifications to the device, by means of control circuitry, allow you to alternatively recirculate or to write new information into the register at will. This is the basis for another simple design project included in the experiment to follow.

EXPERIMENT 18, THE SHIFT REGISTER

Purpose

To show the operation of a shift register and its configuration as a write/recirculate counter.

Materials

1 - 74LS194A universal 4-bit shift register
NOTE - Use a STD 74194 if LS194A unavailable.

Procedure

1. Using a single LS194A bidirectional shift register, hook up the device as shown in Fig. 9-21. A mode control table has been included for convenience. The utility strobe is connected directly to the

clock line, pin 11. The data lines A to D, pins 3 to 7, are left high so that a parallel data load (S0 and S1 = 1) on a positive clock edge will set the register to binary 1111.

2. You should be able to simulate S-to-P and P-to-S conversion, as well as multiply and divide by two, which is nothing more than shift left and shift right.

3. Write/recirculate design problem: In order to design a write/recirculate counter, such as the one in Fig. 9-20, a block diagram including all the desired input and output lines is required. You need to know that the operation will take place with the register in a shift right mode. Draw the block diagram using the following inputs into the control circuitry: serial data in, called data; the write/recirculate control line, which we'll call W; and recirculated data from the SRO output (Q_D), which we'll also call Q_D . The output from the control circuitry must be connected to the SRT input pin on the LS194A. Since this output from the control circuitry is connected directly to the SRT pin, the two signals are essentially one and the same;

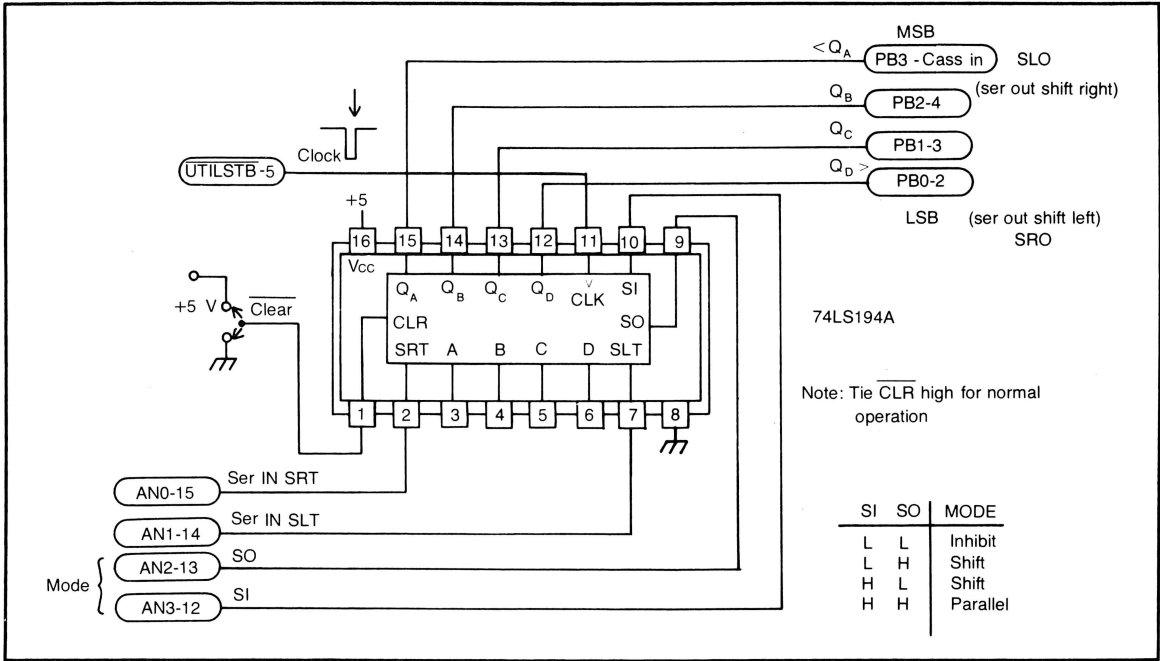


Fig. 9-21. Experiment 18 setup for demonstration of the LS194A.

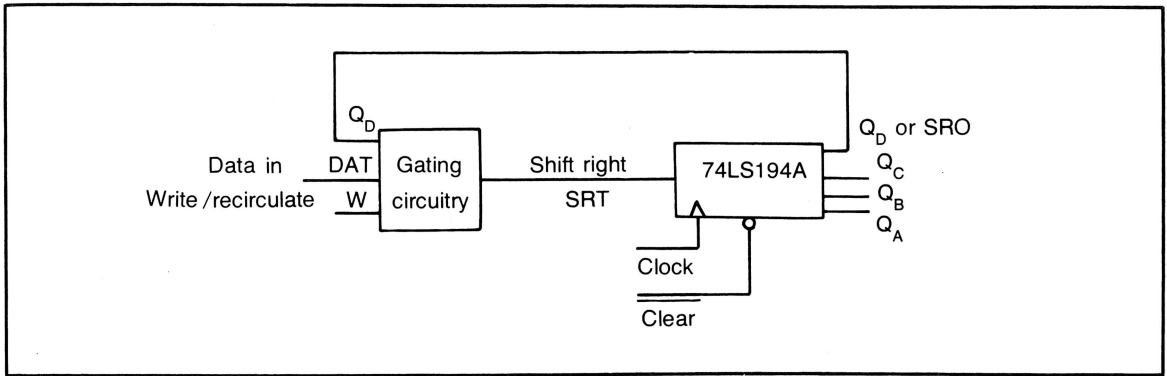


Fig. 9-22. Block diagram statement of design problem in Experiment 18.

therefore, it is appropriate to call the control circuit output SRT also.

4. You should come up with a block diagram looking like that in Fig. 9-22. Now a truth table is in order. Before writing the table, remember that with three inputs into the control circuitry there will be eight lines to the table. The *sense* of the truth table should be as follows. For the sake of argument, let a low on the W line signify a recirculate operation. This means that the SRT line (serial shift right data in) will follow whatever comes out of the Q_D output (SRO). On the other hand, when W is high a write state is signified; then, the output of the control circuitry, SRT will follow the data line.

5. After thinking about the problem, you should come up with a truth table which is essentially the same as that in Fig. 9-23A. You'll notice that there are four minterms in which the output variable STR is true/high/1. Set these down in a four term sum of products equation set equal to SRT. Simplify the equation.

6. Your final equation for the control circuitry should be equivalent to that in Fig. 9-23B. This is a two term SOP equation. You may want to use the double negative rule to come up with the equation in Fig. 9-23C. The reason for this will be clear in a moment.

7. The final circuit with the pin-out is given in Fig. 9-24. After this effort, you can see that the control

circuitry reduced to a single LS00 NAND package with the individual gates connected as shown. The double-negated terms signify the alternative representation for NAND, namely as a negated input OR. This equivalence allows you to use the single NAND IC.

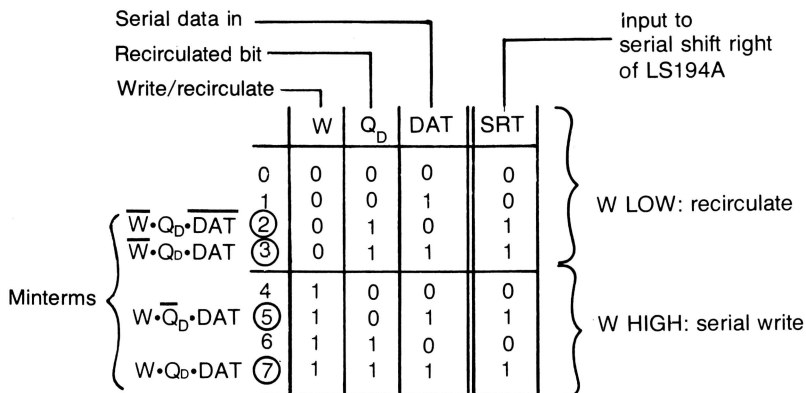
8. After hooking up the circuit, demonstrate for yourself the operation of this write/recirculate device. The mode controls must be set up for shift right operation, with $S_0=1$ and $S_1=0$. You might want to use a single 1 bit and three 0 bits when recirculating data. First clear the register, then serially write in a binary 1, go to recirculate mode, and continuously clock the device. Table 9-2 shows a sample output. Of course, you would do this using only a single line on the BDIS display. Naturally, data can be parallel loaded if you set both S_0 and S_1 to high/1, and clock the data in. Binary 1111 would be loaded if the data inputs lines A to D are left floating high, as in the figure.

Discussion

In the write/recirculate register, you were again shown the usefulness of a little SSI glue to enhance the functions of a flexible MSI device. At the same time you went through the basic combinational SSI design procedure as a review.

The application of this recirculate device as a sequence generator should have been apparent as you clocked the data through the register while in the recirculate mode. Repetitive signals of eight and sixteen bits could be created by using larger

A



$$\overline{m2} + \overline{m3} + m5 + m7 = SRT$$

$$(\overline{W} \cdot \overline{Q_D} \cdot \overline{DAT}) + (\overline{W} \cdot Q_D \cdot \overline{DAT}) + (W \cdot \overline{Q_D} \cdot DAT) + (W \cdot Q_D \cdot DAT) = SRT$$

B

$$(\overline{W} \cdot Q_D) + (W \cdot DAT) = SRT$$

C

$$\overline{(\overline{W} \cdot Q_D)} + \overline{(W \cdot DAT)} = SRT$$

Fig. 9-23. Truth table and resulting sum of products equation for the control circuitry.

shift registers, such as the LS299 and LS670. These are LSI sequential devices, as indicated earlier in Table 9-1. Consult data and application manuals for more information on these latter devices, and for other applications of shift registers.

MSI LATCHES

Before concluding, we'll mention another type of MSI sequential device: the *octal latch*. These may be considered in the same class as the transparent or level-triggered and the edge-triggered D-type

Table 9-2. Sample Output from the Write/Recirculate Register in Experiment 18.

GP SIG:	AN2	AN1	AN0:	PB3	PB2	PB1	PB0
GP IN#:	13	14	15:	CS	4	3	2
LAB L1:	CLR	WR'	DAT:	QA	QB	QC	QD
LAB L2:			:				
<hr/>							
0	0	0	0 :	0	0	0	0
1	1	1	1 :	1	0	0	0
2	1	1	0 :	0	1	0	0
3	1	1	0 :	0	0	1	0
4	1	1	0 :	0	0	0	1
5	1	1	0 :	1	0	0	0

flip-flops discussed in the last chapter, only larger. As you might expect, eight bit latches are quite useful for bus-oriented applications, where temporary storage of an eight data bit word or byte is required. Two such devices in tandem could also store a full sixteen bit address word, typical for today's microcomputers. Naturally, three-state outputs would be necessary for bus isolation. Both transparent and edge-triggered versions have their uses in bus-oriented systems, and octal latches of each type are available.

Two representative and commonly employed octal latches are depicted in Figs. 9-25 and 9-26. The former is the 74LS373, a level-triggered or transparent latch which continuously follows the Data input when the latch enable input is high. Its schematic and pin-out are shown in Fig. 9-25A and B. The LS373 is a 20-pin package which consumes a nominal supply current of 24 mA. It has three modes of operation: follow data input, retain current data (memory), and isolation or high impedance output state. The four line table describing

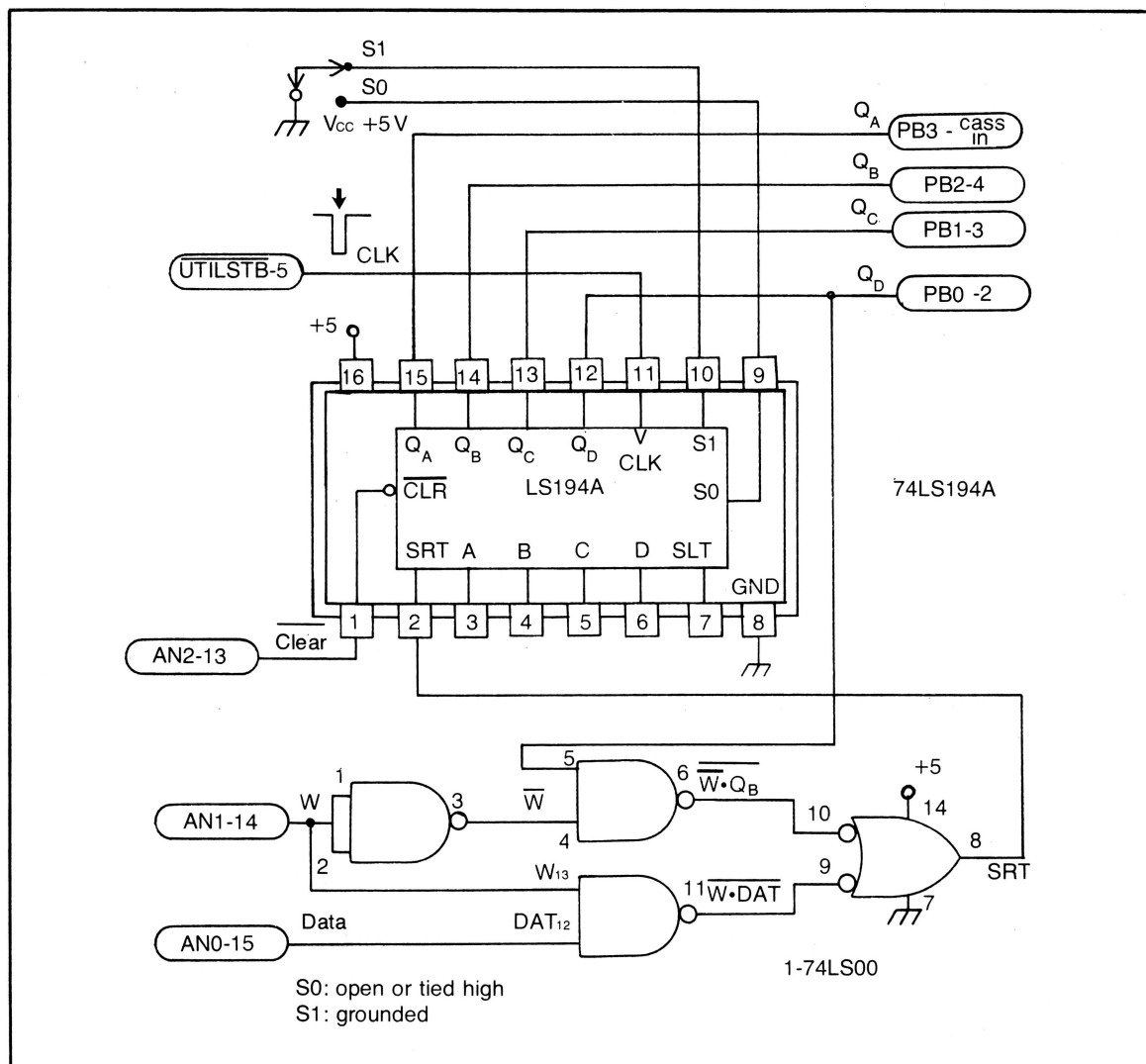


Fig. 9-24. Final control circuit and pin-out, with connection to the LS194A indicated.

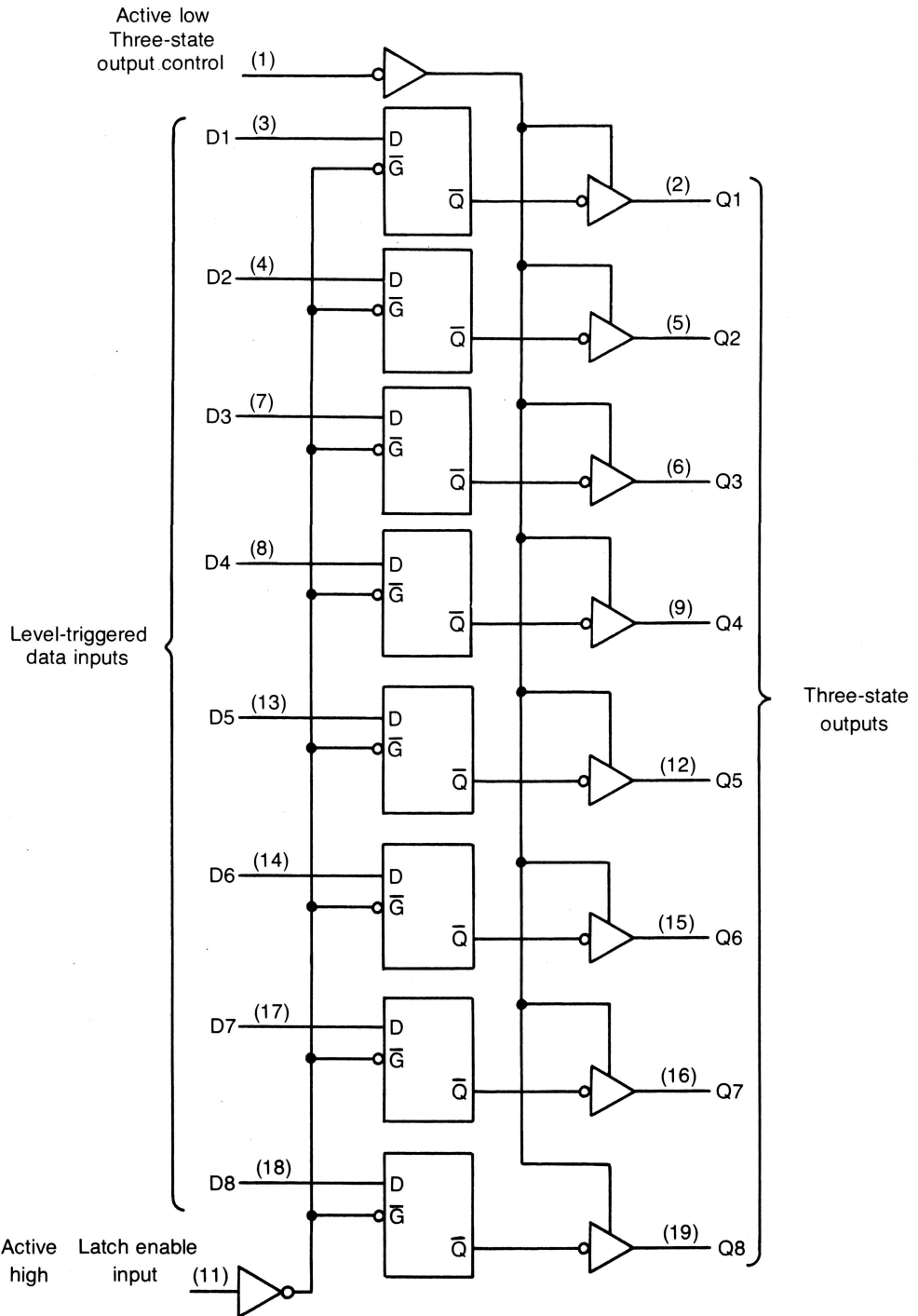
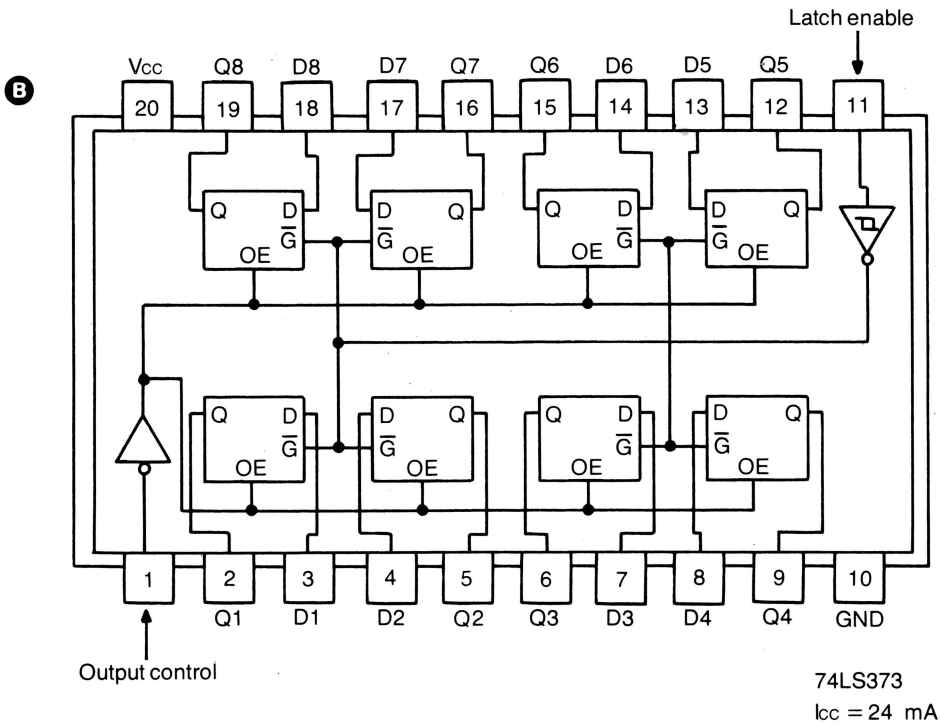
A

Fig. 9-25. A transparent octal latch, the LS373.



C

Output control	Latch enable	Data	Output
L	H	L	L
L	H	H	H
L	L	X	Q0
H	X	X	High Z

Transparent to data

memory

Isolated or high impedance output state

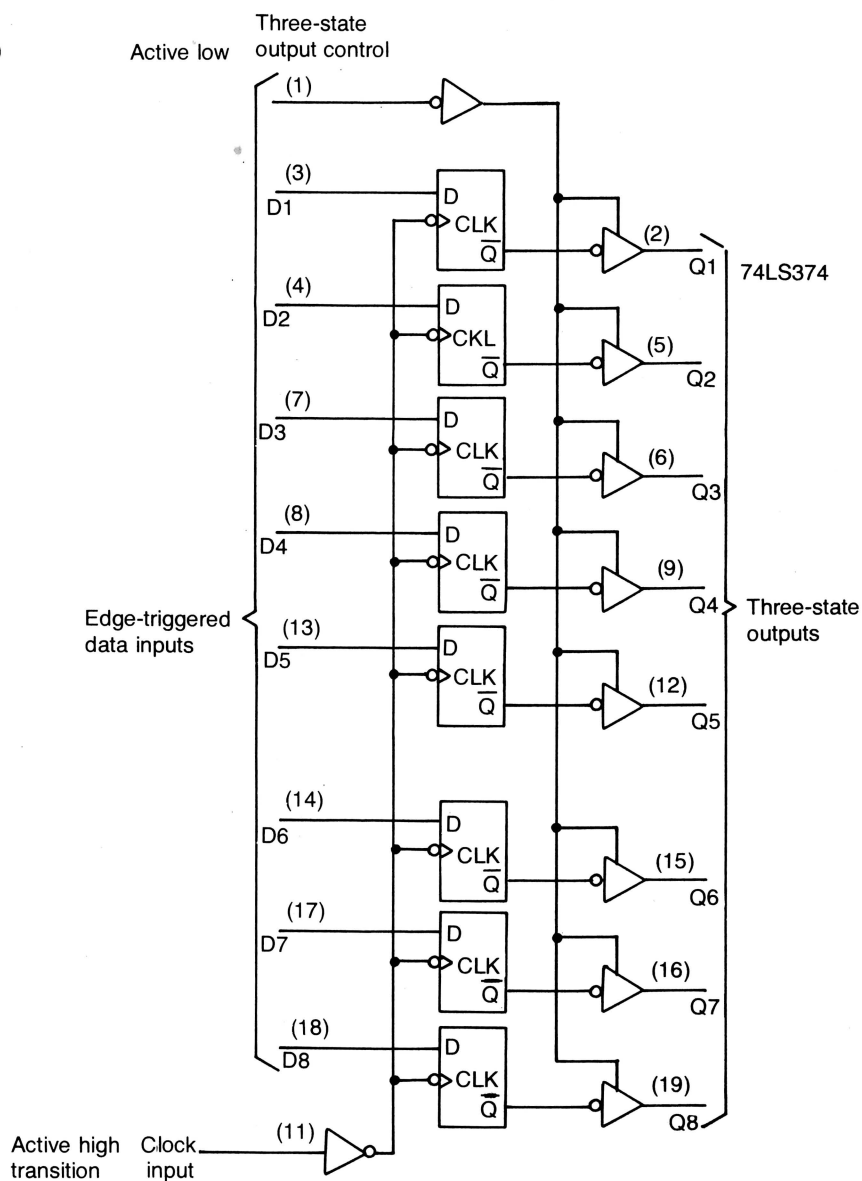
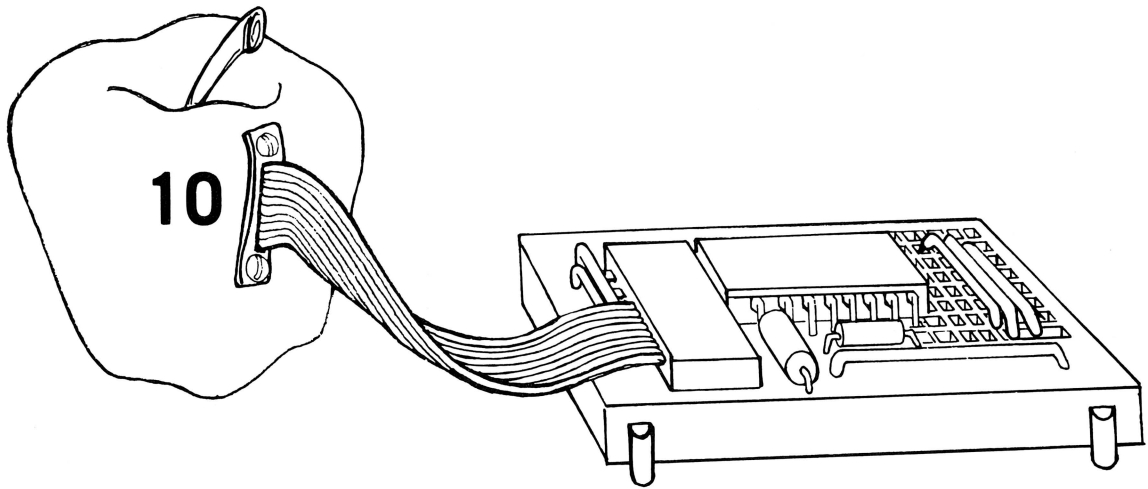
A

Fig. 9-26. An edge-triggered octal latch, the LS374.

the latch's function is given in Fig. 9-25C. Note that the three-state output control is an active low line. It enables the outputs when low and disables them when high.

The other octal latch is the edge-triggered 74LS374. This too is a 20-pin device, consumes about the same supply current (27 mA actually), and has the identical pin-out. The only difference is that

the output state changes on the leading edge of the clock pulse. Even though the state symbol on the flip-flop suggests a down-going trigger, you can see that the clock pulse is inverted first, as indicated in the schematic of Fig. 9-26A. As with the LS373, there is a memory or retain state, and a high impedance output state (isolation from bus line). This is shown in the truth table in Fig. 9-26C.



Combinational MSI Devices

As mentioned in the introductory section on MSI, we will cover the major functional classes of combinational MSI devices in this chapter. Three broad groups of functions will be covered. The first group includes the code conversions and data routing devices, among which are the decoders, encoders, multiplexers and demultiplexers. From each of these four types of combinational MSI functions, one or two representative devices will be detailed, and later demonstrated by experiment. Once you have a handle on a given device, you should have no difficulty in understanding other devices in the same functional class.

Another broad category which we will examine is that of the MSI buffer/drivers, which serve the same role as the SSI buffers covered previously. They are used primarily to increase current drive capability and to provide for bus isolation. The only difference between MSI buffer/drivers and their SSI versions is one of size. Finally, we'll also cover the XOR (exclusive-OR) device and a few of its related functions—error handling, comparison and addition.

CODE CONVERSION AND DATA ROUTING

The easiest way of defining a *decoder* is as a device that converts multibit binary data into decimal data. For instance, three bits of straight binary code can assume a value of 000 to 111, or decimal 0 to 7. Figure 10-1A illustrates such a device which makes the conversion. Each of its eight output lines is assigned a value from 0 to 7, which correspond to the binary input 000 to 111. This device might be called a 3-line to 8-line decoder, or alternatively, a 1 of 8 decoder. In this case, if the input is, say, 101, then output line 5 might go high, while all the other lines remain low (an active high output decoder). Alternatively, output line 5 might instead go low, while all other lines remained high (an active low output decoder).

Four bit binary decoders also exist. They convert four bits of straight binary input into one of sixteen lines of active high or active low output.

BCD decoders, on the other hand, convert four bits of *binary-coded-decimal* data (0000 to 1001) into one of *ten* decimal values (0 to 9). Figure 10-1B shows a 4-line to 10-line decoder. Again, the out-

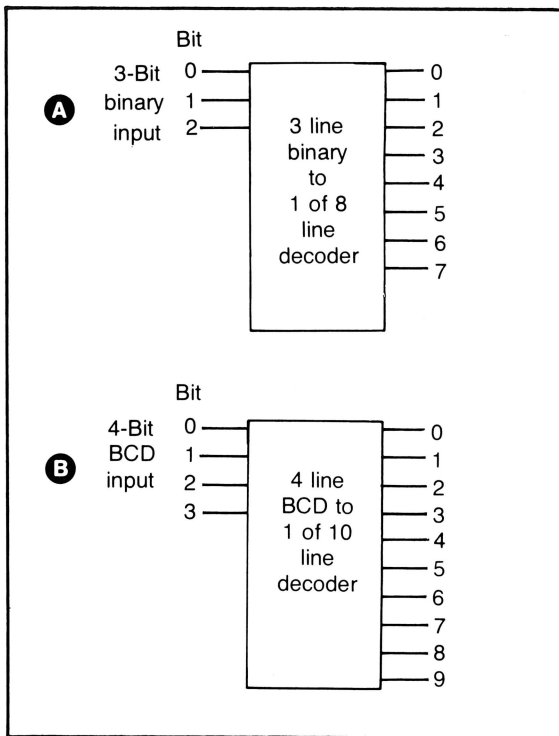


Fig. 10-1. Binary and BCD decoders.

puts are assigned numbers corresponding to the binary input; the output may be either active high or active low, depending on the specific IC.

Another type of decoder is the seven-segment decoder. This device takes four bits of (usually) BCD data, and converts it into an output which will turn on the appropriate segments of a seven-segment LED or LCD (liquid crystal) display. Since seven-segments displays are everywhere—in watches, calculators, automotive displays, test instruments, etc.—it is not surprising that this is a singularly important type of decoder. As suggested in Fig. 10-2, the typical BCD to seven-segment decoder has a few additional control lines which are standard on most such devices. These usually include a line to blank the display for purposes of leading and trailing zero suppression, perhaps a line for enabling the display, and another for turning on all segments in order to test the display.

In regard to the seven-segment display itself, it comes in either of two types: either as a common cathode or as a common anode. In the former, all the cathodes of the LED segments are in common, and are connected to ground. The decoder acts as a

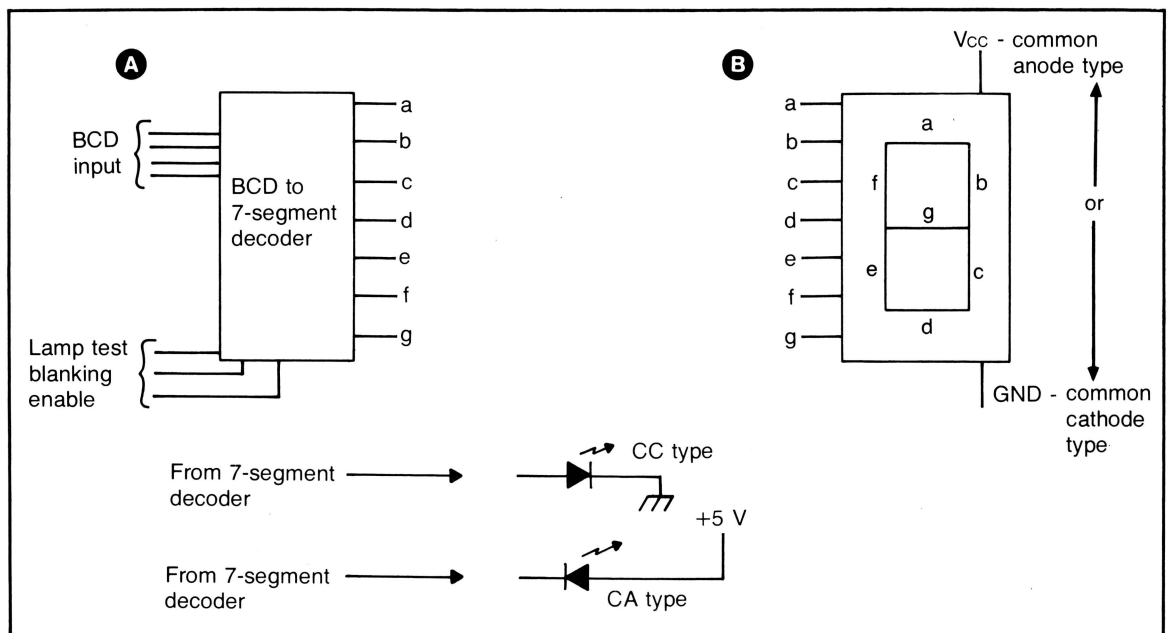


Fig. 10-2. Seven-Segment decoder.

current source in this instance. Common anode displays, on the other hand, have all their anodes connected to Vcc or +5 volts. In this case, the decoder must act as a current sink. Most seven-segment decoders are designed to drive either common cathode or common anode displays, but not both.

Encoders perform the inverse function of decoders. They convert the active signal on one of several input lines into an equivalent binary code at the output. Figure 10-3A shows the 8-line to 3-line encoder, with the output in a straight 3 bit binary format. Figure 10-3B illustrates a 10 line to four line BCD encoder. Outputs may be active high or low, as with decoders.

Specific examples of encoder and decoder chips will be detailed in the next section.

Multiplexers are best thought of as single pole,

multithrow switches in which data on one of *several* input lines is selected and then placed on a single output line. This is illustrated in Fig. 10-4. Here we have an 8-line to 1-line multiplexer, in which the input data line is selected by a three-bit address input, shown in Fig. 10-4A. For example, these data select lines might have a value of 110, in which case the data on input line 6 would be placed on the output. The action can be compared to a mechanical switch, as given in Fig. 10-4B. It is obvious why multiplexers are also called *data selectors*.

Demultiplexers perform the inverse function of multiplexers. They take data off a single input line distribute this data to one of several output lines. Again, the action is much like that of a single-pole multithrow switch, but one which has been turned around from the direction shown in Fig. 10-4B. Shown in Fig. 10-5 is a 1- to 8-line demultiplexer or

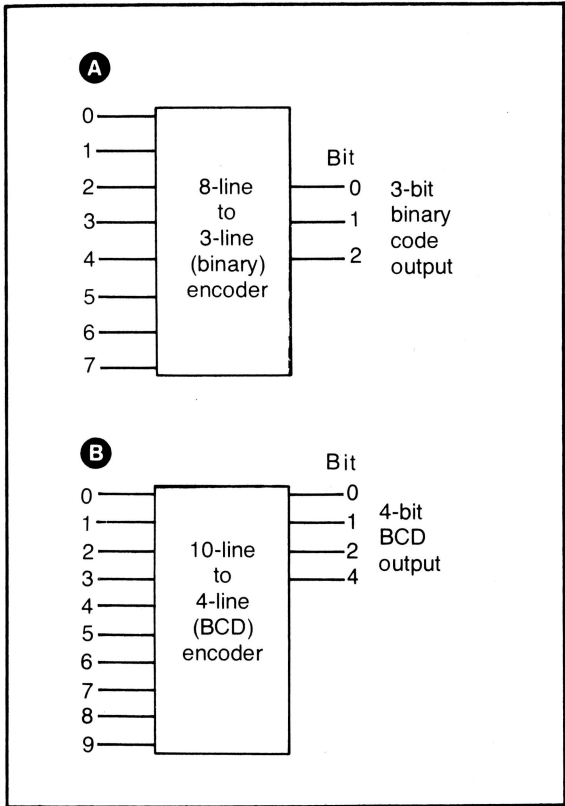


Fig. 10-3. Binary and BCD encoders.

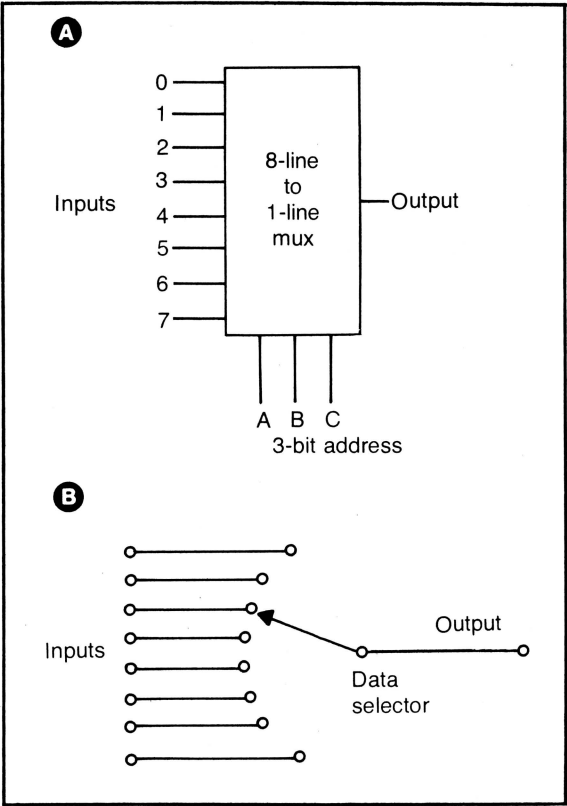


Fig. 10-4. Multiplexers or data selectors are like multipole switches, with several inputs and a single output.

data distributor. As with its multiplexer counterpart, this demultiplexer has a three bit address input, by means of which the input data is distributed to one of the output lines.

Both multiplexers and demultiplexers are important device categories, with the main application being the routing of data within digital systems. Let's now take a look at some representative devices.

Decoders

The representative encoder in Fig. 10-6 is the LS42 BCD (4-line to 10-line) decoder. This is a 16-pin IC which consumes a nominal supply current of 7 mA. Valid BCD data at the inputs results in one of ten output lines going low. This is expressed in the truth table in Fig. 10-6B. Only the active low output states are indicated, for the sake of clarity.

Note that *invalid* inputs in the range from 1010

to 1111 (decimal 10 to 15) do not activate any output lines.

Other examples of decoders are the LS154 4-line to 16-line decoder, and the LS138 3-line to 8-line decoder. The latter device also acts as a demultiplexer, and will be described separately.

The output lines of decoders can be used to selectively activate one of several devices or circuits downstream. Another application is address decoding, related to device selection or activation, but slightly more complex. This is a very important topic which is covered in detail in Chapter 11.

Decoders are also used in display circuits. A high current output BCD decoder, such as the open collector 74145 STD-TTL, is capable of driving incandescent lamps and so called Nixie tube displays. The latter are older numerical displays in which neon tubes, shaped as digits 0 to 9, are lined up one behind the other inside an outer glass bulb. These expensive and power hungry displays have been replaced by multisegment and dot matrix devices for the display of numeric and alphanumeric data. Decoders can also be employed as LED bargraph drivers, as you sometimes see in signal strength indicators (for instance) on some stereo units. Bar graph LEDs are used in some of the demonstration experiments.

Encoders

Figure 10-7 shows a representative 8-line to 3-line encoder, the LS148 cascadable octal priority encoder. This is a 16-pin device with a supply current consumption of about 10 mA.

The LS148 device has several special features that are implied by its title. First, the term *octal* merely refers to the fact that there are eight inputs; each of these is active low. The binary data outputs of this device are also active low.

(Important note: This means that an output of 010 would actually represent decimal 5. Likewise, binary 101 would be decimal 2.)

By *priority* encoder, we mean a device which gives preference to the highest-valued input line which is active. For example, if two of the active low inputs are grounded, say line 3 and line 5, then the active low output would register a value of 010.

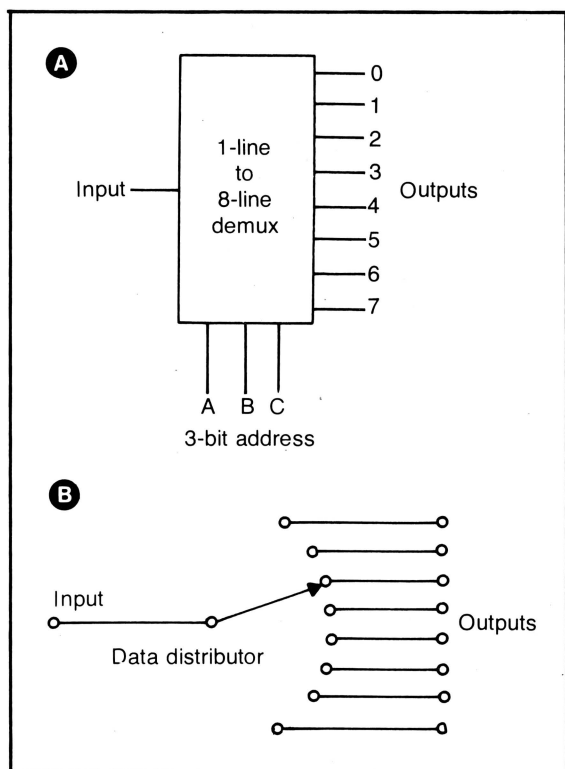
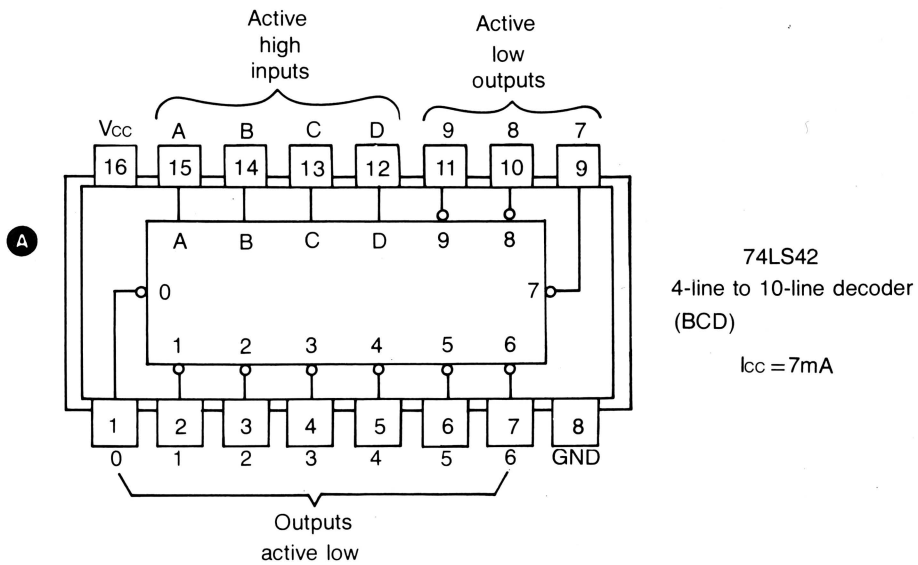


Fig. 10-5. Demultiplexers or data distributors are also like multipole switches, but with one input and several outputs.



B

Decimal	Inputs				Outputs									
	D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	L	L	L	L	L									
1	L	L	L	H		L								
2	L	L	H	L			L							
3	L	L	H	H				L						
4	L	H	L	L					L					
5	L	H	L	H						L				
6	L	H	H	L							L			
7	L	H	H	H								L		
8	H	L	L	L									L	
9	H	L	L	H										L
Invalid inputs	H	L	H	L	All outputs high									
	H	L	H	H										
	H	H	L	L										
	H	H	L	H										
	H	H	H	L										
	H	H	H	H										

Blank outputs are high

Fig. 10-6. Pin-out and truth table of a 4 to 10 line BCD decoder, the LS42.

That is, the device is designed so that it never becomes confused when two or more input lines are simultaneously active. In a practical situation, the highest-valued input may not be the one actually intended. However, this arbitration in favor of the highest input avoids arbitrary logic levels at the outputs, and so is a desirable feature.

If you refer to the truth table, Fig. 10-7B, you'll note a few other lines. One is the EI, or

enable input, line. It is an active low input, and permits normal operation when in this state. When high, the encoder is disabled, and all outputs are high (inactive) as indicted in the table.

The other lines are the EO (enable output) and GS (gate select) lines. These are active low output lines which allow the LS48 to be *cascaded* or expanded into larger encoder circuits. In order to explain this cascadable feature of the LS148, we

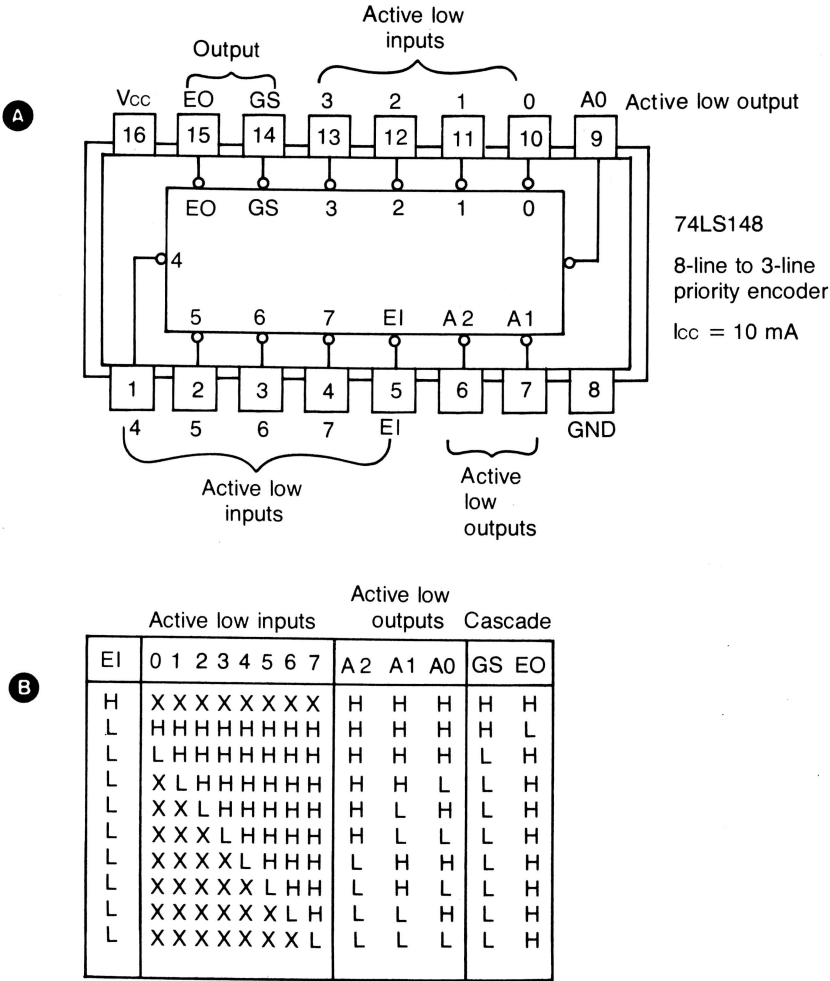


Fig. 10-7. Pin-out and truth table of an 8 to 3 line priority encoder the LS148.

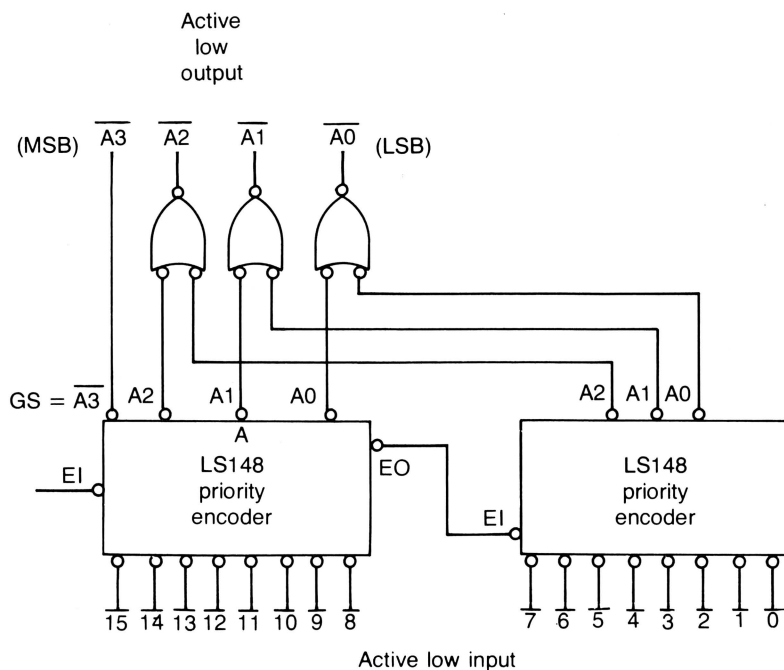


Fig. 10-8. For a 16 to 3 line encoder, the LS148 can be cascaded.

have to refer to Fig. 10-8. (Assume that for the circuit shown the EI line is low, so that the circuit is enabled).

Suppose you needed a 16-line to 4-line encoder. When two LS148 encoders are connected as shown in Fig. 10-8, then the device to the left (device A) will encode for all inputs in the range of decimal 8 through 15. Its GS (gate select) output will be active whenever one (or more) of its input lines is active. This means that A3, the most significant (most leftward) bit of the four bit binary output, will be active. The exact value of the output will then be reflected by bits A2, A1, and A0. The binary output will then fall in the range 0111 (decimal 8) through 0000 (decimal 15), using the active low convention for the LS148.

If, on the other hand, only one of the lines 0 through 7 is active, then GS/A3 will remain inactive (high). The binary output will then fall in the range of binary 1111 (decimal 0) through 1000 (decimal 7). The three AND gates are necessary to register the

values of the three lower bits A0 - A2. They are represented in negative logic convention in order to emphasize the active low character of the outputs.

Two common applications of encoders are as components in keyboard logic and in interrupt circuitry.

You can imagine that if sixteen single-pole single-throw switches (with one side connected to ground) were attached to the 16-line priority encoder just described, you would effectively have a small keypad. Ten keys could be assigned for decimal input, one for enter, the rest for cursor movement or other special functions. If attached to a computer, this simple keypad would have to be read by the appropriate software, of course.

Keyboard encoding circuitry is more complex than this in anything larger than a 16-key keypad. When you require a typewriter sized keyboard with a full ASCII character set, it is best to use one of the dedicated LSI decoding chips currently available.

The priority encoder is also used in computer

interrupt applications. Each computer has one or more input control lines which, when activated by some outside device, interrupt whatever the computer is doing. Control is then turned over to a special program that the user or someone else may have written. This *interrupt handler* program decides which of several interrupt lines has been activated, and then it in turn gives control over to one of several other programs. After the task of *servicing* the peripheral which interrupted the computer is completed, then the computer resumes whatever it was doing previously. A simple example follows.

For instance, a factory microcomputer may be monitoring air temperature for climate control in the plant, security devices, fire alarms, status of certain machinery, etc. During this time, it may also be performing its primary task, such as giving the current inventory status (quantity, price, source, etc.) to any of several users. However, the occurrence of an emergent event will necessitate the interruption of this multiuser inventory function so that the computer can initiate appropriate action—sounding alarms, activating sprinklers, turning off machinery or cutting power, sending preprogrammed phone messages to the proper authorities, etc. Now, if several such events occur, then a further breakdown in order of importance is assigned. For instance, a fire alert may take importance over the overflow of some production bin. It is not necessary to do all this work in software, however. You can, through the use of priority interrupt circuitry, assign priority to certain events over others and save a bit on code.

Obviously, the priority encoder can play a key role in such circuitry. It provides hardware arbitration between potentially conflicting interrupt signals for computers that have several interrupt input lines. Naturally, it is up to the user to assign the proper priority to the various peripherals.

Multiplexers

The LS251 is as good an example as any of an MSI multiplexer chip. This 16-pin DIP consumes a nominal 7 mA of supply current. It has the useful feature of complementary three-state outputs for bus oriented applications.

Figure 10-9A illustrates the pin-out of the LS251. The truth table in Fig. 10-9B defines the operation. When S' (the three state output control line) is low, the Y output follows the selected data line (1 of 8). At the same time, the W output follows the inverse of the data. In other words, W is the complementary output. When S' is high, these two outputs are isolated, or in other words in the high impedance state.

Obviously, multiplexers can be used to select one of several data sources for transmission on a single data line, or on either of two complementary lines in the case of the LS251. This is desirable in instances where serial data transmission is employed. If you were monitoring several serial data sources, you would save on hardware by using a multiplexer; all you would need would be some software to scan each input line. This could be done by periodically incrementing the three select lines, and reading the serial data.

This principle can be extended to *bused* data transmission. You need only provide one multiplexer for each line on the data bus. Eight LS251s would allow eight parallel data peripherals to share the same system data bus. In this case, the multiplexer functions as a parallel data router.

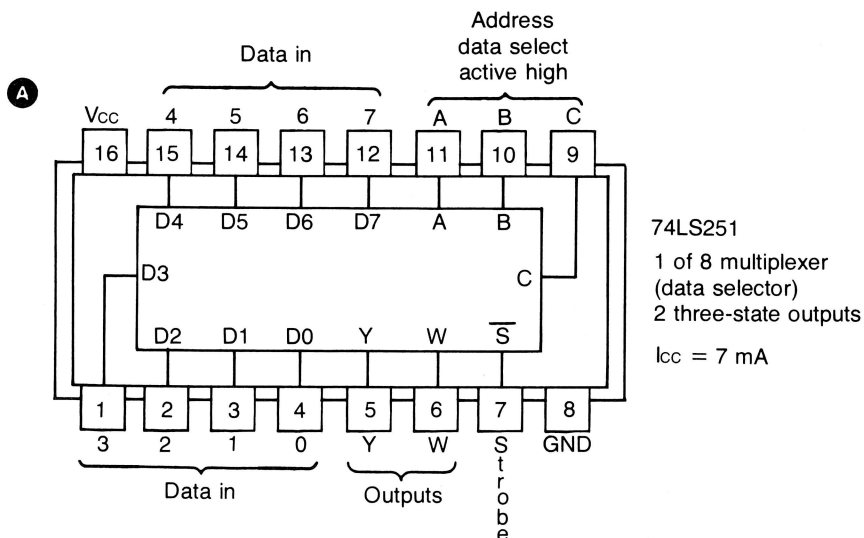
Multiplexers are also used in the encoding circuitry for larger keyboards of 32 keys or more.

Another application is in multidigit displays. Multiplexers economize on the amount of display circuitry needed by allowing each digit in the display to share the common BCD to 7-segment components in the circuit. Such systems do require clocking, but for a six or seven digit display the savings in space, power and hardware for the decoding elements is considerable.

Demultiplexers

We'll talk here of a very popular device, shown in Fig. 10-10. This is the LS138, a 16-pin package using 6 to 7 mA of supply current. Actually, this device can be used as either a decoder or demultiplexer.

When used as a decoder, the output will be active low. When operating as a decoder, the three enabling or gating inputs— $G1$, $G2a'$, and $G2b'$ —



B

Inputs		Complementary outputs		
Strobe or enable	Address or select	Y	W	
\overline{S}	C B A	Y	W	High impedance output
H	X X X	Z	\overline{Z}	
L	L L L	D0	$\overline{D0}$	
L	L L H	D1	$\overline{D1}$	
L	L H L	D2	$\overline{D2}$	
L	L H H	D3	$\overline{D3}$	
L	H L L	D4	$\overline{D4}$	
L	H L H	D5	$\overline{D5}$	
L	H H L	D6	$\overline{D6}$	
L	H H H	D7	$\overline{D7}$	

Fig. 10-9. Pin-out and truth table of a 1 of 8 line multiplexer, the LS251.

must be at the proper logic level. G1 must be high, and the other two, sometimes referred to simply as G2, must be low. When so enabled, the three bit binary input—A, B, C—will set one of eight outputs low, leaving the others high. This is indicated in the truth table of Fig. 10-10. (The highs are left blank in this part of the table for clarity).

How does the device operate as a multiplexer? By using one of the enable lines as a data input!

Usually G2 (either 2a or 2b) is used as a data line, because in this way the selected output will be noninverted. To see how this works, imagine you've selected line Y4 as the output to which you want to *distribute* your data. The data select lines would be HLL (100) in the truth table. When G2 is low, Y4 is low, and when G2 is high, Y4 is high. This is indicated by the arrows in the figure.

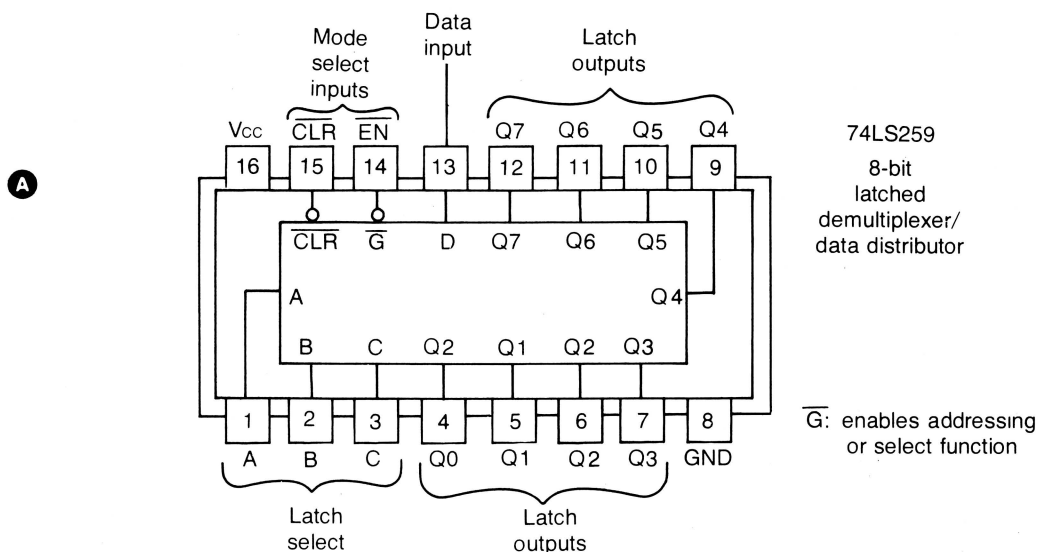
Of course, if you needed the additional feature

The data routing function of data distribution is performed by the LS138 when it is used as a demultiplexer. A more important application is in address decoding.

There are variants of the above four basic

types of combinational functions. One is the LS259 addressable latch. Essentially, it is a demultiplexer with some added functions that can be selected by means of mode settings. The key feature is that each output line is attached to a flip-flop. This permits data to be retained in those outputs which are not currently being addressed. The LS259 is a 16-pin IC with a nominal current demand of 22 mA; this is more than the other devices mentioned, but acceptable in view of the flexibility of this IC. The

Fig. 10-10. Pin-out and truth table of a 3 to 8 line demultiplexer/decoder, the LS138.



B

	Mode select inputs		Output of addressed latch	Output of other latches	Description of mode
	Clear	\overline{G}			
1	L	L	D	L	1 of 8 multiplexer
2	L	H	L	L	clear all latches
3	H	L	D	Q0	"addressable latch"
4	H	H	Q0	Q0	memory

← Addressed latch follows data in (D) all others are low

← Same as (1) but other latches are unchanged

C

Latch select inputs			Output
C	B	A	Latch addressed
L	L	L	Q0
L	L	H	Q1
L	H	L	Q2
L	H	H	Q3
H	L	L	Q4
H	L	H	Q5
H	H	L	Q6
H	H	H	Q7

Multiplexer and addressable latch modes

Fig. 10-11. Pin-out, operational modes and truth table for the LS259 addressable latch.

pin-out and truth table descriptions of this device are given in Fig. 10-11.

Four modes of operation are possible with the LS259, and they are determined by the settings on the clear and G (enable) inputs.

- Clear and G = low: It can act as a 1 of 8-line demultiplexer, wherein input data is distributed to one of eight output lines. In this mode, the addressed output follows the data input, and all other (unaddressed) outputs are low.
- Clear = low, G = high: All outputs are cleared or set low.
- Clear = high, G = low: Again, the LS259 acts as a 1 of 8 demultiplexer, with the addressed outputs following the data line. But in this mode,

the unaddressed outputs retain their prior state. This is the addressable latch mode of operation.

□ Clear and G = high: The device is disabled in this mode. In this memory state, all latches (output lines) remain in the state they were in just prior to disabling, regardless of the condition of the select inputs A, B, and C.

The applications of the addressable latch are probably obvious to you. It is convenient to be able to distribute data to one of several output lines and have that data retained, that is, latched on those lines. The logic levels on the output lines could represent control signals that enable/disable other devices. In the next chapter, you'll see how the LS259 plays a key role in Apple's game port circuitry.

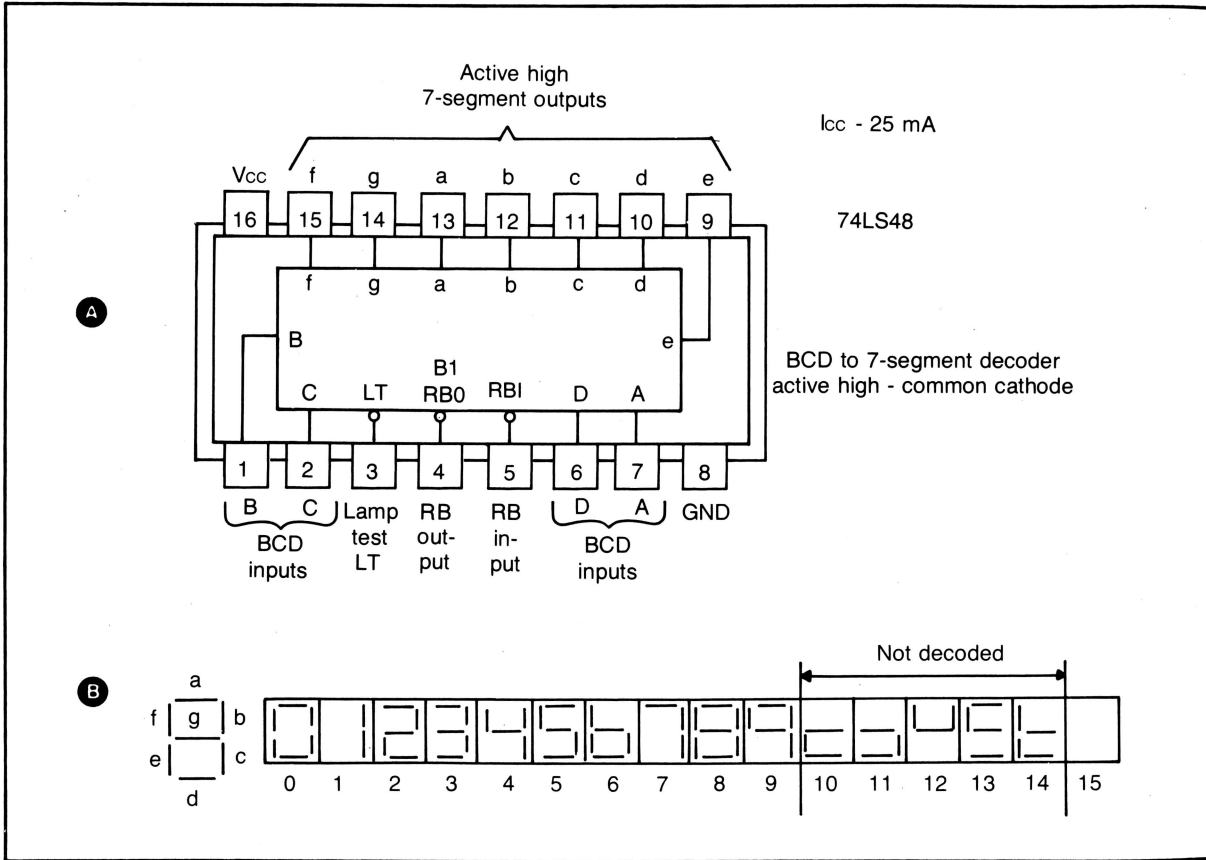


Fig. 10-12. Pin-out, seven-segment display, and truth table for the LS48 common cathode decoder/driver.

The Seven-Segment Decoder/Driver

This is a slightly more complex class of devices than the five just covered. As the name suggests, these devices are used to decode BCD inputs into seven segment code so as to light up the appropriate segments in a numeric display. Further, they also *drive* the display; that is, they provide the necessary current to light up the LED.

A typical device is the LS48 BCD to 7-segment decoder with active high outputs. See Fig. 10-12. It will provide sufficient current sourcing capability to drive the average small to medium sized 7-segment LED. The current requirement of this 16-pin IC is about 25 mA with outputs open. More current is consumed when the 7-segment display digit is connected, about 6 mA per lit segment.

Each of the segments of a 7-segment display

are labeled a through g, as shown in Fig. 10-12B. The truth table that describes the seven segment output code for each 4-bit input is given in Fig. 10-12C. One thing to observe is that the decimal digits 0 to 9 are registered (in seven segment code) just as you would expect. However, the invalid inputs 1010 to 1110 produce strange-looking patterns on the seven-segment display. An input of 1111 blanks the display (all outputs are low). Ideally all invalid inputs should result in a blanked display. That is, these inputs should be fully decoded, but the additional circuitry was not designed into the LS48 to do this. However, this is usually not a drawback, because in practice illegal values never reach the decoder if the display circuit is designed properly.

You'll note that there are two additional inputs

C

Inputs				Outputs								$\overline{\text{BI}}/\text{RBO}$	Function or decimal value
LT	RBI	D	C B A	a	b	c	d	e	f	g			
L	X	X	X X X	H	H	H	H	H	H	H	H	H	Lamp test
H	L	L	L L L	L	L	L	L	L	L	L	L	L	Blanking
H	H	L	L L L L	H	H	H	H	H	H	H	H	H	0
	X	L	L L L H		H	H						H	1
		L	L L H L		H		H	H		H		H	2
		L	L L H H	H	H	H	H				H	H	3
		L	H L L L		H	H				H	H	H	4
		L	H L L H	H		H	H		H	H	H	H	5
		L	H H L L				H	H	H	H	H	H	6
		L	H H H H	H	H	H						H	7
		H	L L L L	H	H	H	H	H	H	H	H	H	8
		H	L L L H	H	H	H				H	H	H	9
		H	L L H L				H	H			H	H	10 A
		H	L L H H			H	H				H	H	11 B
		H	H L L L		H					H	H	H	12 C
		H	H L L H	H			H		H	H	H	H	13 D
		H	H H L L				H	H	H	H	H	H	14 E
		H	H H H H									L	15 F

Blank spaces are low

1010-1110 (10-14) invalid but not decoded

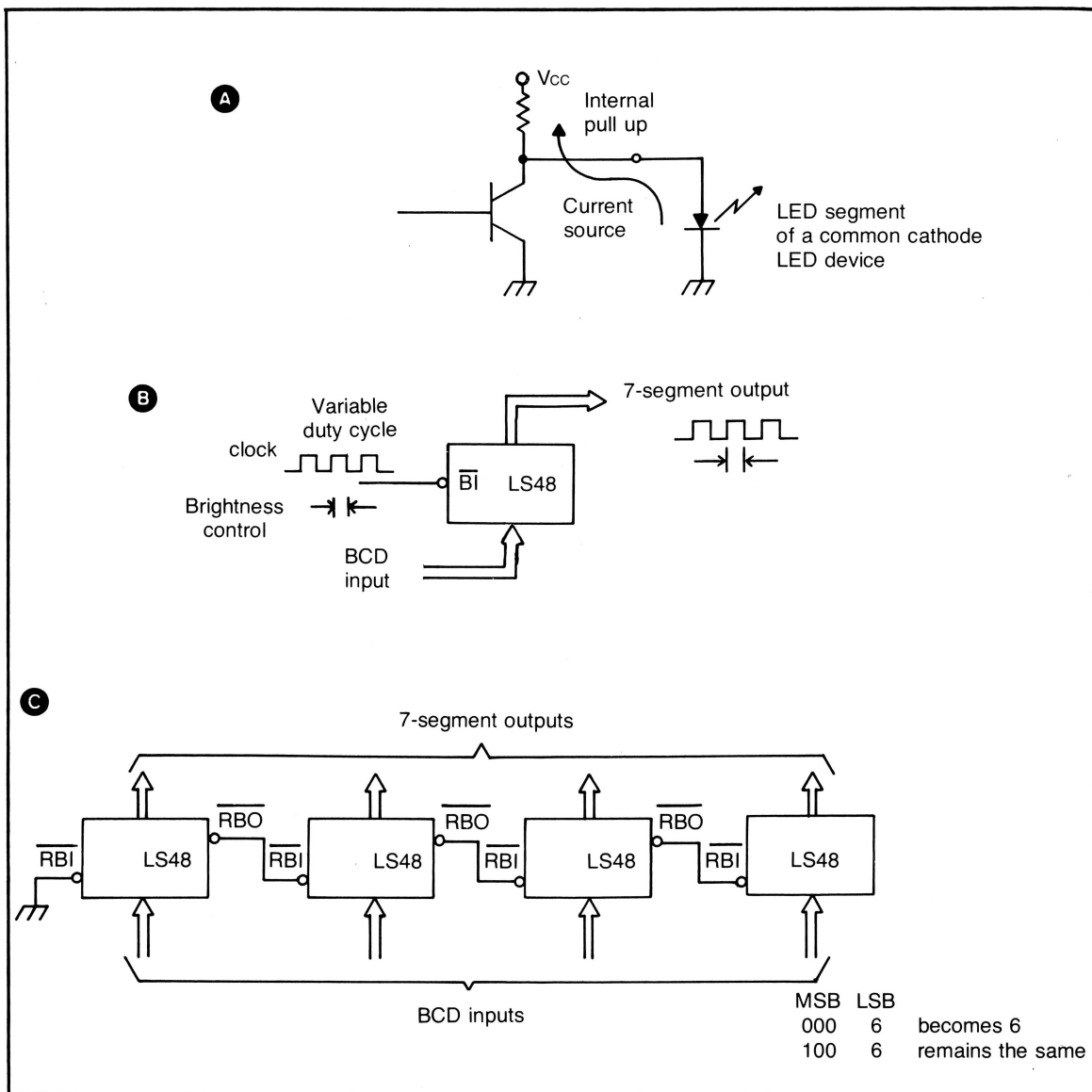


Fig. 10-13. (A) Current source logic of the common cathode driver. (B) Use of variable duty cycle clock to control brightness. (C) Cascading LS48s for multidigit display with leading zero blanking.

and one other output besides the BCD lines. One of these inputs is the lamp test or LT' . When low (active) it will turn all outputs high and thereby lights up all seven segments. When LT' is low, it makes no difference what the other inputs are, as it overrides them all. The other input is the active low, ripple blanking input, or RBI' . Provided that

LT' is inactive (high) and the BCD input is 0000, the RBI' input will make all outputs low. This is a *blanking* condition.

The additional output is the ripple blanking output, RBO' , also an active low line. This line is high during a blanking condition, and when 1111 is placed on the inputs. This output can, if desired,

also be used as a direct blanking input as well. This is why it is sometimes referred to as the BI'/RBO' line.

It's important to understand how the LS48 drives an LED segment. For this, and in order to make some sense of these extra input and output lines, look at Fig. 10-13.

Figure 10-13A shows how the active high output line off the LS48 drives an LED segment. The output transistor acts as a current source with respect to the load presented by the LED. This diode segment is connected with its cathode grounded, as are all other diode segments in the seven segment package. This is why such displays are called common cathode devices. Note that the output is not a totem-pole but rather an open collector configuration. More accurately, the pull-up resistor has been included inside the IC package! This is especially convenient, and is why this particular decoder has been chosen both for this discussion and for a later experiment.

Now, what about those ripple blanking lines? They can be understood with reference to two applications. First, the BI'/RBO' line can be used directly as a blanking input, as just mentioned. Specifically, when BI' is low the display blanks off. If you were to feed a sufficiently high frequency square wave to this input and could vary its duty cycle, you would have a way of varying the brightness of the display. See Fig. 10-13B. This can be accomplished for one or several displays simultaneously. You would need a clock source of about 1 kHz or so for this purpose. (Too low a frequency would result in display flicker).

As far as the purpose of RBI' and RBO' are concerned, refer to Fig. 10-13C. The most significant digit of this four digit display is decoded by the leftmost LS48 IC. If you were to input a number 6, for example, the display might actually read 0006. This is cumbersome to read. You want some way of suppressing the leading zeros. This is accomplished by grounding the RBI' line of the most significant decoder, and cascading the RBI' and RBO' lines as shown in the figure. Then whenever there is a leading zero input (0685, 0020, 0006, etc.) the corresponding zero digit will be blanked or

suppressed. This follows directly from the truth table. Note that a number like 1006 will be registered as such, with the two middle zeros displayed.

Trailing zero suppression is also possible with the LS48, with some circuit modification (not shown). This means that a number such as 4.7000 would be displayed as 4.7 while a number like 4.7001 would be displayed as is.

Other Devices

There is a tremendous variety of display chips available. Among the seven-segment decoder/drivers are open collector devices, such as the LS47 common anode and LS49 common cathode ICs. High power chips for larger LEDs or incandescent displays are also available. Other types of displays incorporate multiple digits and the necessary decoding circuitry all in one package. This approach is excellent for clock modules and test equipment.

LED and LCD displays with more than seven segments exist; the extra segments permit display of letters as well as numbers. As you would expect, there is a line of decoder-driver ICs for this family of displays. More prevalent are the dot matrix displays, often in a 5×7 arrangement, much like the screen matrix for text display on your Apple. Virtually any upper or lower case figure can be represented using these alphanumeric displays and their associated decoder chips. In particular, Hewlett Packard makes a variety of such devices. (See their *Optoelectronics Designers' Catalog*. Write to Hewlett Packard, 640 Page Mill Road, Palo Alto, Calif. 94304.)

EXPERIMENTS FOR MSI COMBINATIONAL FUNCTIONS

The organization of this set of experiments is somewhat different from those of the preceding chapters. First, the express purpose of the experiments can be stated here: to demonstrate the basic operation of several representative devices by means of simple circuits. As a matter of both convenience and visual appeal, three of the demonstrations incorporate LED and bar graph displays.

Only one annunciator is used, namely for enabling/disabling the device through the EI input. All PB lines are used—three to indicate the encoded binary output, and one for the EO (enable output) status. The LED indicator is connected to show GS (gate select) status. Verify the operation, with reference to the truth table in Fig. 10-7B. Observe that the outputs are in active low/negative logic, so that 010 actually represents 101 in positive logic, or decimal 5. 101 would be equivalent to decimal 2, etc.

As an additional project, you may want to hook up the circuit in Fig. 10-8, which is a full 16 line encoder. You are cascading the EO and EI lines in this circuit, and adding $\frac{3}{4}$ of an LS00 NAND package. A second 8-switch DIP-switch will be necessary. As shown, you need only the four PB lines to indicate the encoded output.

EXPERIMENT 20, BINARY DECODER/DEMULPLEXER

Materials

- 1 74LS138 Decoder/Demultiplexer
- 1 LED bar graph (R.S. 275-082 or equiv.)
- 1 330 ohm resistor

NOTE: You can use eight discrete LEDs, with their anodes tied in common to +5 volts through a 330 ohm resistor in place of the bar graph shown. The bar graph looks neater and is more compact.

Figure 10-15 gives the demonstration circuit for the LS138 3-line to 8-line decoder/demultiplexer. Four annunciators are used, three for the 3 bit input, as an enable (gating) line to G2A'. First show how the device operates as a decoder, by first enabling the device with a low/0 on the active low G2A' pin. With the configuration shown, the top-most bar is decimal 0 (binary 000) and the bottom-most is decimal 7 (binary 111). The convention is that when an LED segment in the bar graph is lit, then the logic level on the corresponding output pin is low/0. This follows from the truth table. (The LEDs are driven by output-low/current-sink logic, which you've seen many times before).

Now operate the device as a demultiplexer. Given that an output low is indicated by a lit LED, you can reason that by toggling the G2A' input to high, you are outputting a logic high. The LED will be off, naturally. That is, you've addressed a given output and are toggling it on and off with G2A', which acts like the data line in a demultiplexer. Address a different output line, and you can do the

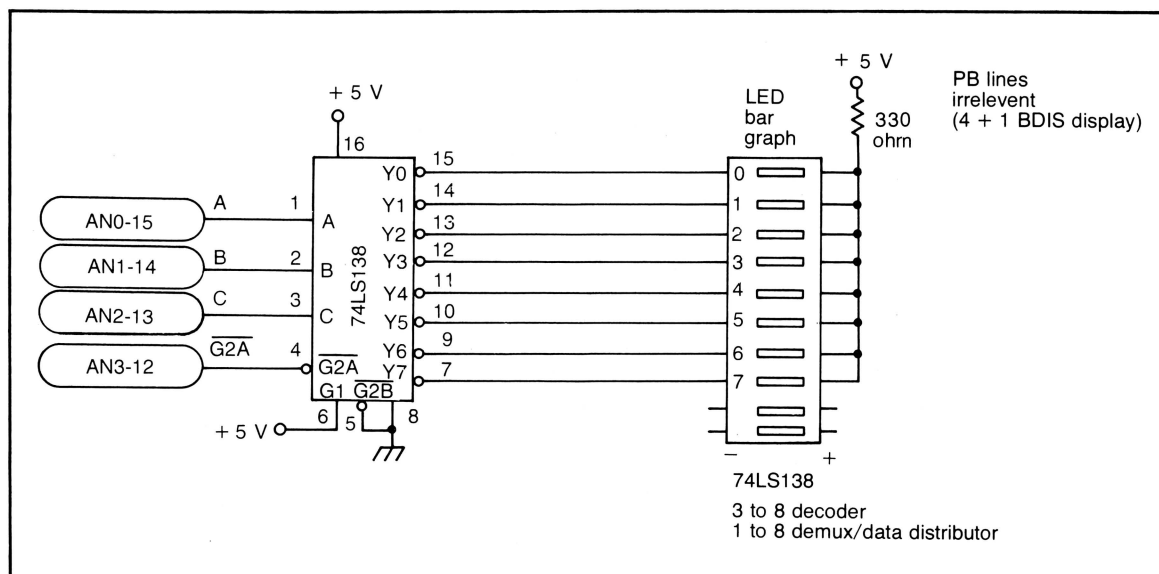


Fig. 10-15. Experiment 20. LS138 decoder/demultiplexer circuit.

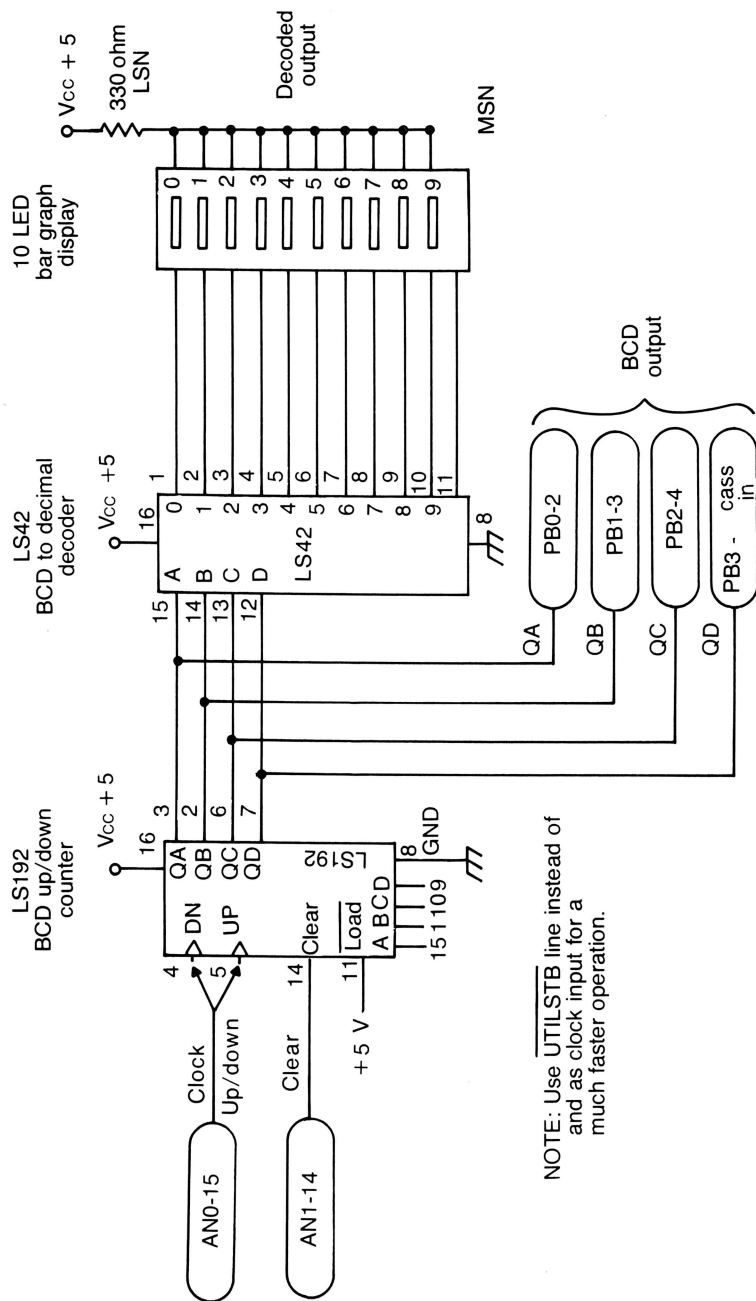


Fig. 10-16. Experiment 21. LS42 BCD decoder with counter and bar graph.

same thing. You've demonstrated the data distribution or demultiplexing function of the LS138.

If you had used the G1 line and the data line, the output would be the inverse of the level on G1. This follows from the truth table in Fig. 10-10.

EXPERIMENT 21, BCD DECODER

Materials

- 1 74LS192 BCD Counter
- 1 74LS42 BCD Decoder (4-line to 10-line)
- 1 LED Bar Graph
- 1 330 ohm resistor

The LS42 BCD decoder is another active low output device. Again, we can use a bar graph display (or 10 discrete LEDs if you wish), driven by current sink logic.

Figure 10-16 shows the setup for this experiment. A BCD counter is used to enhance the demonstration of the BCD decoder. Two annunciator lines are used: one to clock the count up or count down pins as desired, and the other as a clear input. Leave the data lines A-D open. The counter outputs are monitored by four pushbutton lines. You need

only add the two extra connections to the bar graph to complete the display. Zero is at the top and nine is at the bottom of the display for the configuration shown.

If you wish to operate the circuit forward, connect AN0 to the up input. To see the lighting sequence in reverse, clock the down input.

For faster operation, replace the AN0 line with the utility strobe. The speed of the lighting sequence will increase several fold.

Do not disassemble this experiment.

EXPERIMENT 22, SEVEN-SEGMENT DECODER/DRIVER

Materials

- 1 Circuit setup from the prior experiment
- 1 74LS48 Common Cathode 7-Segment Decoder/Driver
- 1 Common Cathode 7-Segment LED Display—use any available display, the R.S. 276-067 (MAN84A) for example.
- 1 LED
- 1 330 ohm resistor

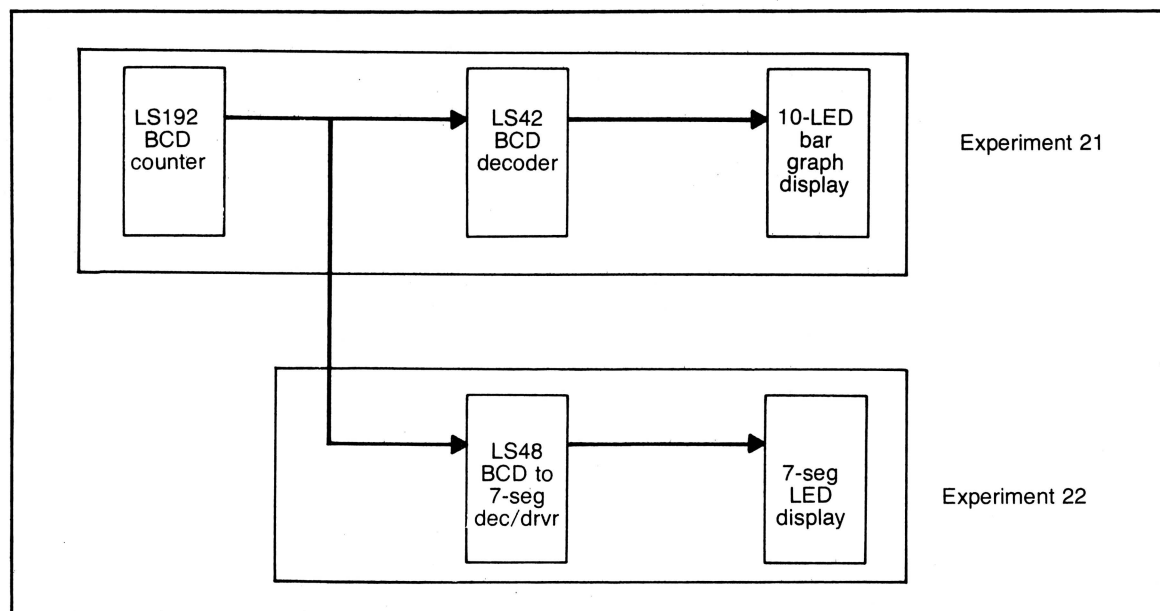
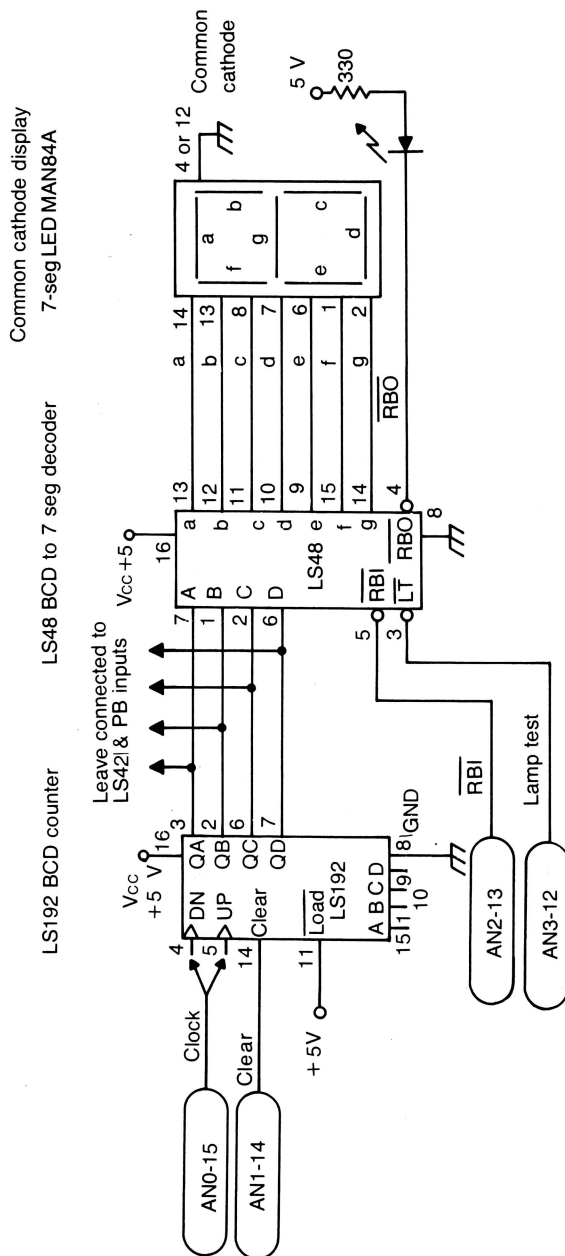


Fig. 10-17. General setup for Experiment 22.



NOTE: use UTILSTB line instead of AN0 as clock input for a much faster operation

NOTE: 7-seg LED used in MAN84A. substitute others, but be sure to use common cathode devices. Pinouts vary.

Fig. 10-18. Experiment 22. LS48 circuit.

The configuration for our last circuit is given in Fig. 10-17. A portion of it consists of the entire circuit from Experiment 21. You need only add the LS48 device and the seven-segment display.

The detail of the circuit is given in Fig. 10-18. Two additional annunciator lines are connected to the RBI' and LT' inputs of the LS48. A single LED is used to indicate the status of the RBO' line.

Set up the BDIS display as shown in Table 10-1. In Table 10-1A the RBI line is left high (inactive) so that the zero state will be shown on the seven segment display. If RBI' is set low, as in Table 10-1B, then the display will be blanked on zero. When operating this circuit, you can see how the clocked input of the LS192 translates into three outputs: BCD code on the BDIS display, a one of ten line output on the bar graph, and a number on the seven segment LED. Verify the operation of the lamp TEST, RBI' and RBO' lines as well. Use the utility strobe for faster operation if you wish.

THREE STATE DEVICES

Figure 10-19 illustrates two octal buffer/

drivers, the LS240 and LS241. These three state devices consist of eight buffers connected as two groups of four. A separate enable input controls the output of each group. Data is buffered as eight bits, that is, each device can buffer a full byte of data. Further, the three state outputs permit isolation for bus oriented applications.

The LS240, inverts the data. The LS241, on the other hand, is a noninverting octal buffer. Each is a 20-pin DIP. Both devices have essentially the same pin-outs. The only difference (aside from inversion and noninversion) is the active state of the G2 enable pin. In the LS240 it is active low. In the LS241 it is active high.

Each device boosts the current sinking capability to 24 mA. This is three times as much as the normal LSTTL sink capability of 8 mA. Also, these devices can source current (output high current I_{OH}) providing a nominal value of 15 mA of source current if desired. Of course, current consumption increases to about 30 mA for these devices.

These two octal buffers are *one-way*. This means that they can pass data in only one direction.

Table 10-1. Setup of BDIS Display for Experiment 22.

A	GPSIG:	AN3	AN2	AN1	AN0:	PB3	PB2	PB1	PB0
	GPIN#:	12	13	14	15:	CS	4	3	2
	LABL1:	LT'	RBI	CLR	CLK:	QD	QC	QB	QA
	LABL2:				:				

	0	1	1	0	0 :	0	0	0	0
B	GPSIG:	AN3	AN2	AN1	AN0:	PB3	PB2	PB1	PB0
	GPIN#:	12	13	14	15:	CS	4	3	2
	LABL1:	LT'	RBI	CLR	CLK:	QD	QC	QB	QA
	LABL2:				:				

	0	1	0	0	0 :	0	0	0	0

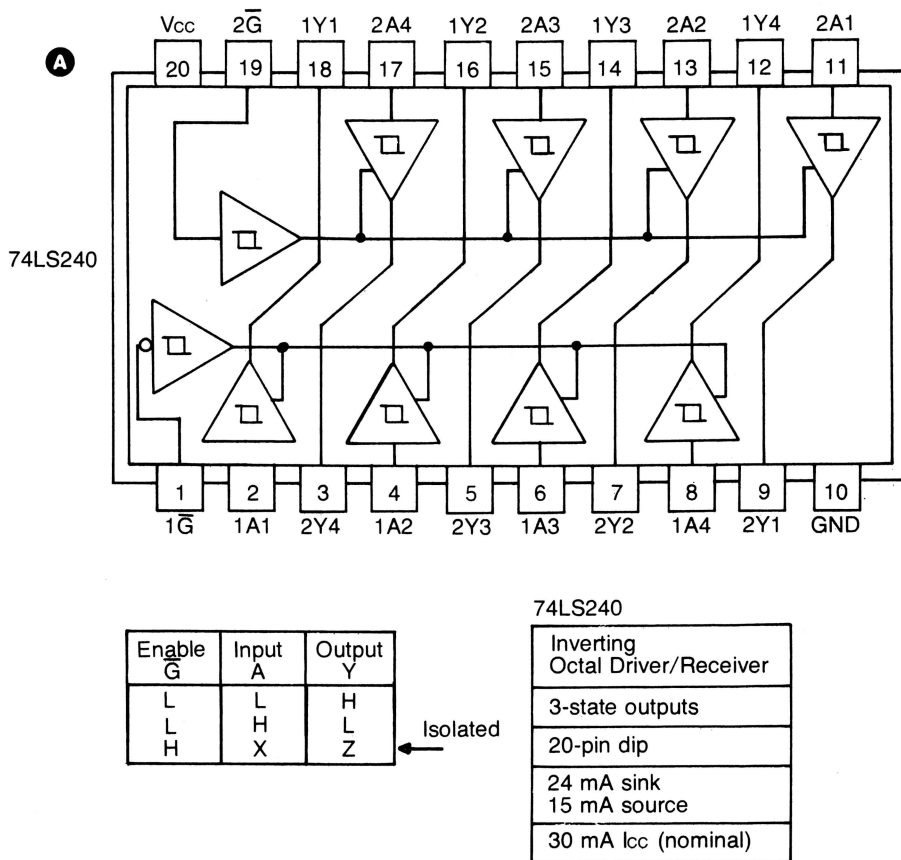


Fig. 10-19. Octal buffer/drivers (bus drivers!).

For bidirectional data bus applications, you would need two-way devices. These would be able to send or receive data in either direction under the influence of some control signal. These bus transceivers (as opposed to one-way bus drivers) are commonly employed in computer systems. One such popular device is the LS245 octal bus transceiver shown in Fig. 10-20.

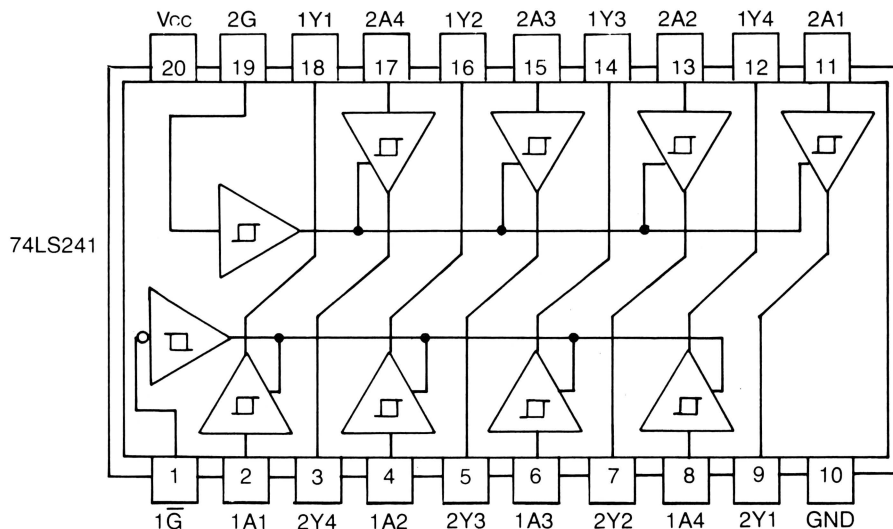
This device is a noninverting transceiver with a single active low enable pin (\bar{G}), and a direction control pin (DIR). When \bar{G} is high, the outputs are in high impedance state, when low, the device acts as a transceiver. Direction control pin DIR sends data from pin A to B when it is high. When DIR is

low, data passes in the reverse direction from B to A.

Sink and source capability is similar to the LS240/241 type device. But since there are twice as many buffers in the LS245, supply current demands is about 60 mA.

Figure 10-21 illustrates the role of the transceiver in a computer system. The READ/WRITE control pin on some microprocessor chips (the Apple's 6502 for example) is used to control the direction of data flow. When high, the microprocessor is executing a read operation. Peeking data from memory would be an example. In such a case, data is going into the chip, from A to B as illustrated.

B



Enable \bar{G}	Input A	Output Y
L	L	L
L	H	X
H	X	Z

← Isolated

74LS241

Noninverting Octal driver/receiver
3-state outputs
20-pin dip
24 mA sink 15 mA source
30 mA I_{CC} (nominal)

The reverse is the case for a write operation. Here, DIR would be low, and data would be sent in the B to A direction—to RAM memory or to some peripheral.

EXCLUSIVE-OR FUNCTIONS

Our discussion of combinational MSI devices would not be complete without mention of the exclusive-OR function. The representative chip, the LS86, is just on the other side of the SSI/MSI division of device complexity. You were introduced to the XOR in Chapter 4 but its function was not detailed at that time.

First, refer to Fig. 10-22 to review XOR oper-

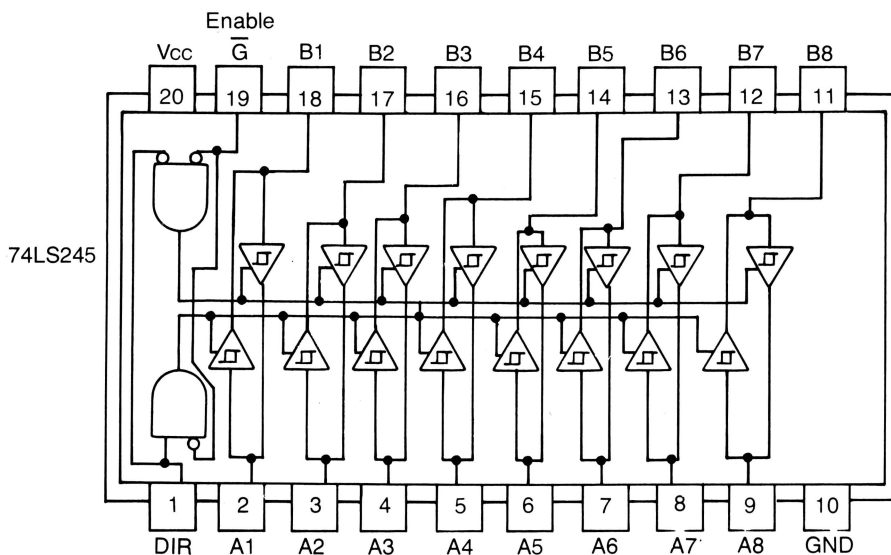
ation. The LS86 performs a combinational function such that whenever the two inputs are different the output is high/1. When the two inputs are the same, the output is low/0. The encircled plus sign signifies the XOR function. The schematic symbol is given in the pin-out.

As to the LS86 itself, it is a 14-pin DIP with a very modest power requirement of about 6 mA.

What about the applications of this device? There are many uses, but for brevity we'll look at three major ones here: data comparison, error handling, and addition.

Comparison of Digital Words

Comparison of multibit data is often required



Enable G	Direction control DIR	Operation
L	L	B data to A
L	H	A data to B
H	X	Both A and B isolated

74LS245
Noninverting octal transceiver
3-state outputs
20-pin dip
14 mA sink 15 mA source
60 mA I _{CC} (nominal)

Fig. 10-20. An octal bus transceiver.

in digital systems. This can be done in hardware, so that the relationship between two digital words of equal length (4, 8, 16, or more bits) can be specified. If we call the words A and B, then the possible relationships are: $A > B$; $A = B$; $A < B$. The circuit shown in Fig. 10-23 indicates only whether two four-bit words are equal or unequal. The circuit is designed to indicate a high when the words (nibbles) are equal. Only if all outputs of the XOR gates are low will the output of the total circuit be high. This final output is taken off a four input

NOR gate. Note that if the LED lights up (output is low) a mismatch is signaled.

Conversely, if you wanted to indicate a match by an output low, rather than a high, a four input OR gate would be used, as in Fig. 10-24A. Here, a lit LED indicated agreement, bit for bit, between the two nibbles. Or, you could use the output arrangement in Fig. 10-24B so that you could use either form of match/mismatch indication.

More complex, dedicated comparators exist. But the principle of those using XOR logic as es-

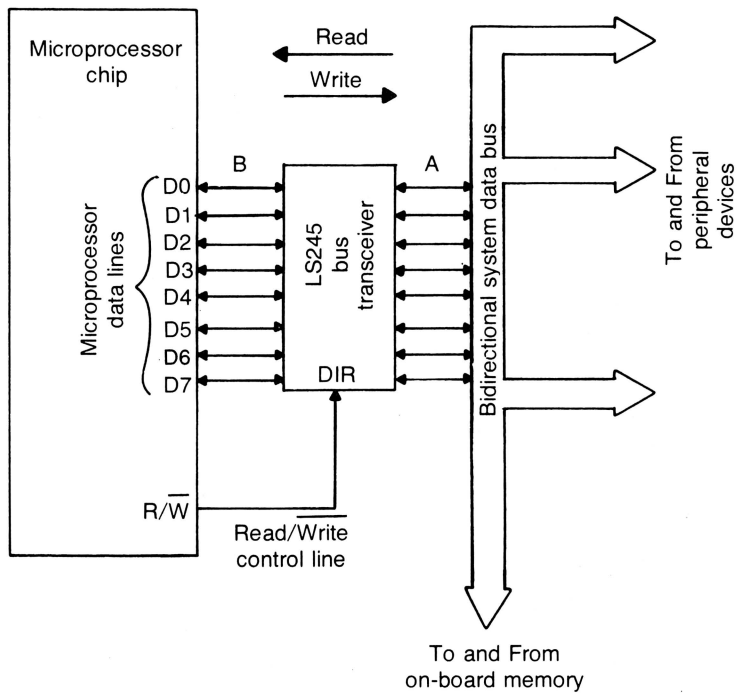


Fig. 10-21. Application of a transceiver in a computer system.

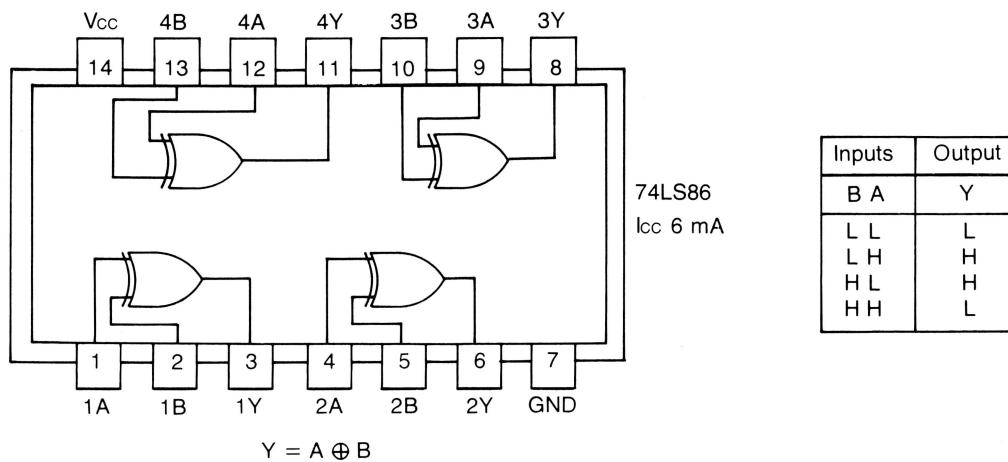


Fig. 10-22. The LS86 XOR device.

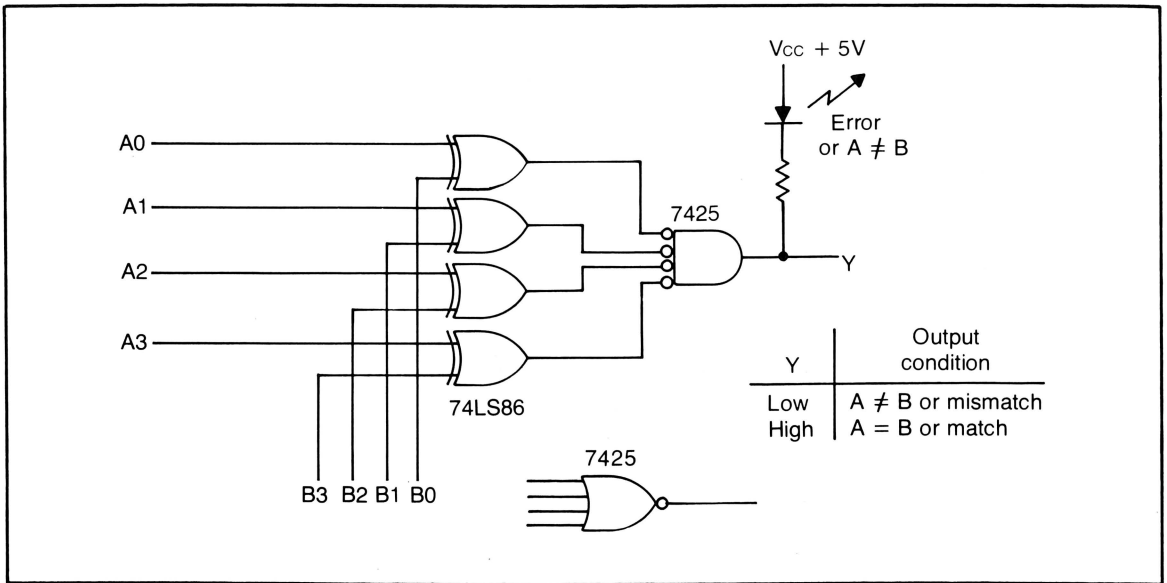


Fig. 10-23. Application of XOR as comparator with high = match.

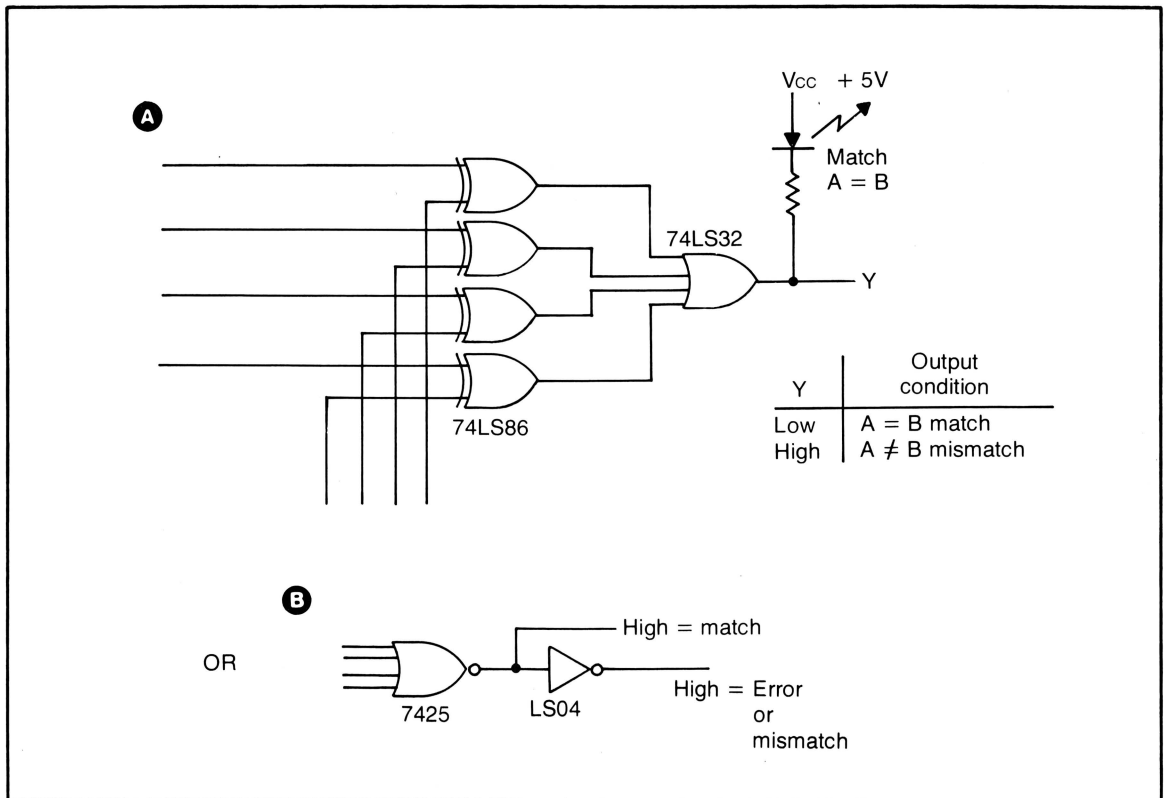


Fig. 10-24. (A) Application of XOR as comparator with low = match. (B) Combined outputs.

entially the same as that illustrated here.

Parity Checkers/Generators

A gross way to indicate whether or not there is an error in a digital word is to know whether it contains an even or odd number of binary ones. If you could somehow include this information in the transmitted word, you would have a crude means of indicating whether or not a single bit error had occurred. Parity is the term referring to whether there are an even or odd number of binary ones in a given digital word.

Specifically, the parity concept implies that an extra bit is added to the word so that the number of ones is always even or odd, depending on whether you are using an even or odd parity convention. This extra bit is called the parity bit.

To make this concept clear, we'll take a six bit word as an example. There are two conventions for parity. Referring to Fig. 10-25A, you can see that the same six bit word has an extra bit added so that the total number of ones is either even or odd. Even parity and odd parity are conventions. Once you

adapt one form or parity in a system, you must stick with it. As you can see from the example:

- The odd parity system would demand that the total number of ones, including the parity bit, is odd.
- The even parity system demands that the total number of ones, including the parity bit, is even.

Figure 10-25B gives another example of even and odd parity applied to a different six bit word.

You may ask if there is any inherent advantage of one parity convention over the other. The answer is provided in Fig. 10-25C. The zero value word in Fig. 10-25C, 000000, would have for its parity bit another zero if even parity were used; it would have a one for the parity bit if the odd parity convention were used. Since it is usually desirable to have at least one high bit in a given word, the odd parity system would be preferred.

Regarding the circuit implementations for generating and checking parity bits, refer to Fig. 10-26.

In Fig. 10-26A, a four gate parity generator circuit is shown. It consists of $\frac{3}{4}$ of an LS86 plus an inverter. The circuit checks for an even or odd number of ones and yields the appropriate parity bit for both even and odd parity conventions. You should verify the logic of this circuit yourself. Once the parity bit is generated, it can be "tacked on" as an extra bit before the digital word is sent out on the bus or transmission line.

Figure 10-26B illustrates a parity checker. A receiving module may have such circuitry built in to check for errors before passing it on to the rest of the system. The circuit shown operates as an odd parity checker. If there are an even number of ones in the data portion of the word, then a 0 will be output from gate 3. The parity bit, which should be 1, is then XORed with this 0 to give a final output of binary 1. This output high indicates no error. A low would indicate an error and light up the LED.

Finally, you may wonder about two errors that cancel each other out, so that no error is detected. This would occur if a 0 was somehow transformed in transmission to a 1, and vice versa for another bit.

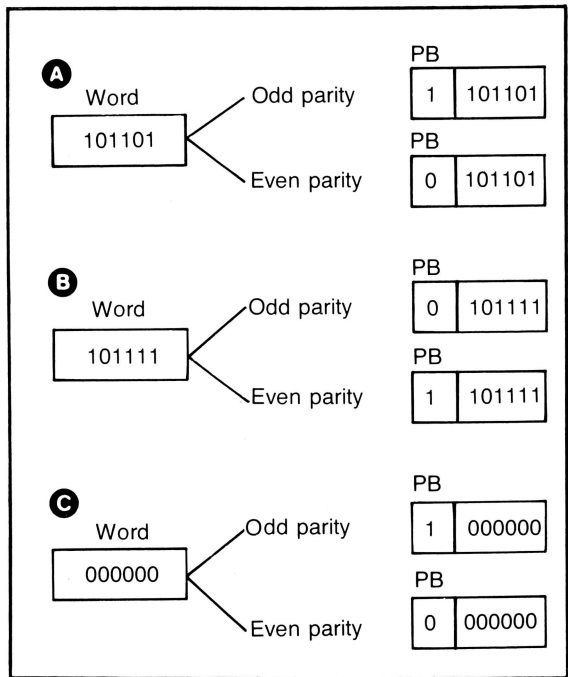


Fig. 10-25. Odd and even parity conventions.

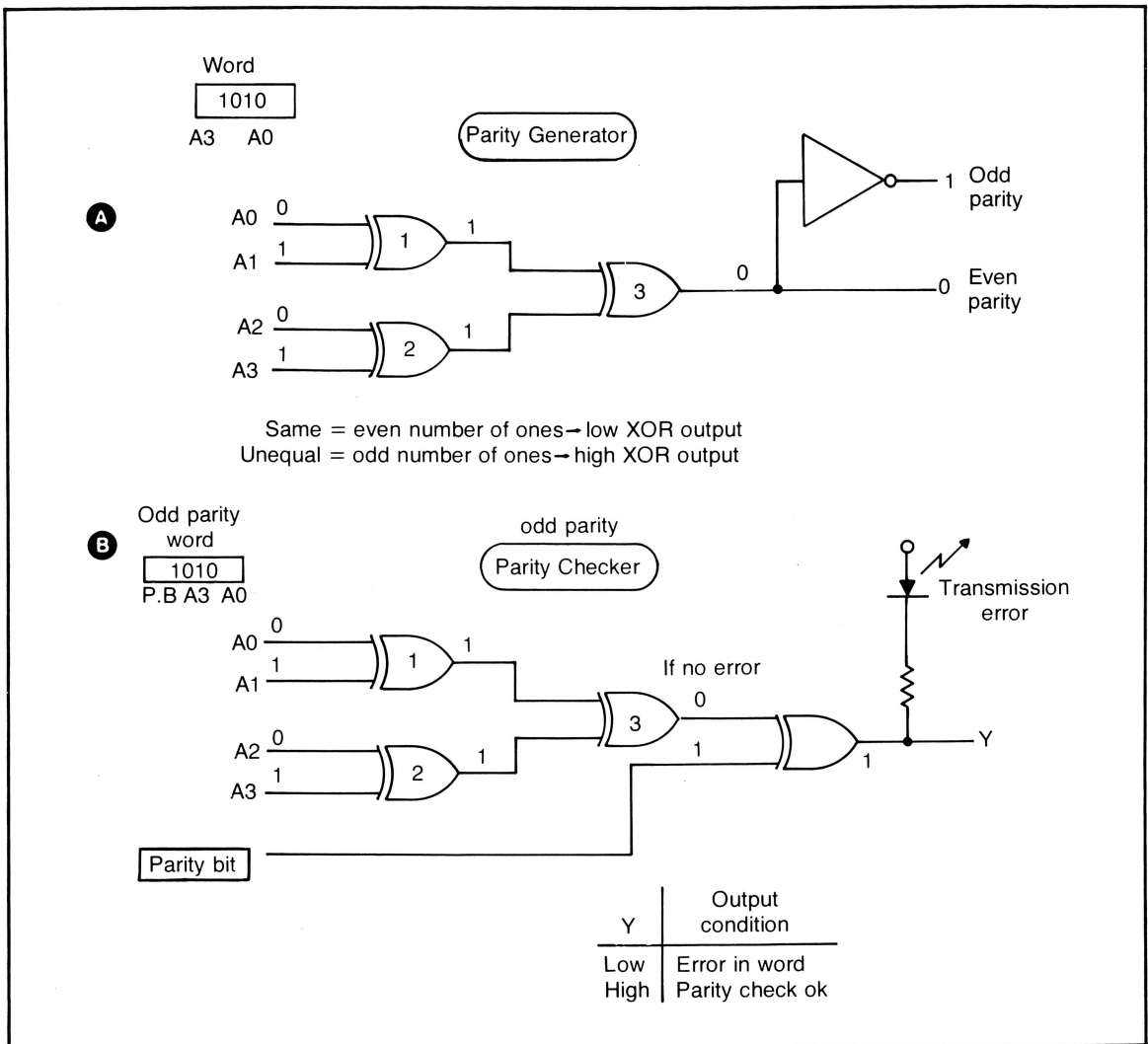


Fig. 10-26. (A) Parity generator - odd and even. (B) Odd parity checker.

This is an obvious limitation on the single parity bit system. Also, what about the possibility of the parity bit itself being erroneously transmitted (through static interference, noise “glitches”, etc.)? In this instance, a false error would be indicated. The parity checker would see an error when there really was none.

What is the solution to these limitations on the single parity bit system? The answer is in part provided by multiple parity bit systems; these perform sophisticated error detection, and some of

them will correct errors as well.

One way to detect multiple errors is the so-called vertical and horizontal system. For example, data is sometimes transmitted in blocks, as a series of digital words. Parity bits for these blocks can be generated and later checked in two ways: horizontally, word by word; and vertically, column by column. This gives a reliable check of the overall parity of the entire block of data. Multiple errors can be checked for. If errors do occur in transmission, the block is simply retransmitted. This is a

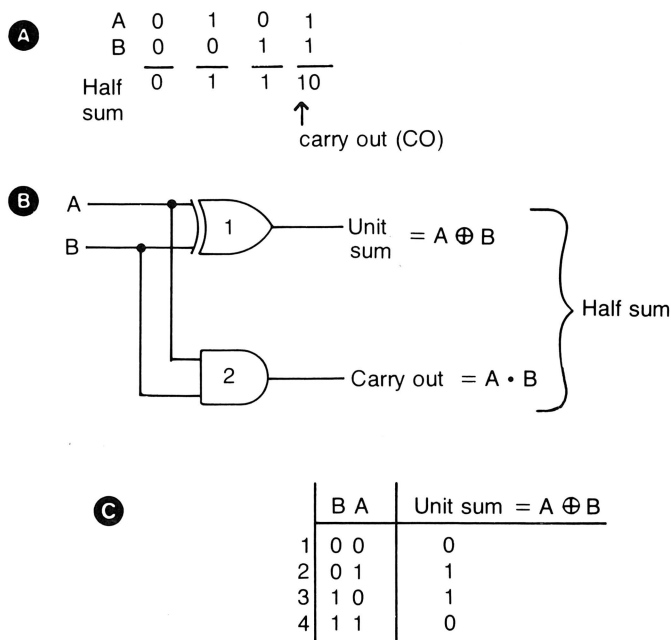


Fig. 10-27. Half-adder using XOR.

very cheap form of error detection and correction, and does help overcome the problem of multibit errors.

More sophisticated detection/correction schemes do exist, such as the Hamming Code method, in which multiple bits permit the actual correction of errors as they occur. There are also methods of transmitting data in a matrix format so that correction of up to three consecutive errors is possible! Obviously, these schemes are more complex than the simple go/no-go and retransmit method mentioned above. However, they are useful in one-way applications, as in telemetry (automated transmission of data) where retransmission on demand may not be possible.

Adders

XOR devices can perform binary addition. To see how an XOR gate does this, refer to Fig. 10-27A. The addition of two bits is illustrated. As you can see, there are only four possibilities, the last

one having a carry out (CO) bit of one. In Fig. 10-27B is the XOR implementation of this addition. The XOR gate performs the actual addition of the two bits, which one might call the unit sum or place sum; this is indicated in tabular form in 10-27C. The AND gate registers the CO bit.

The entire circuit is called a *half adder* circuit because one element is missing: the carry in bit (CI) from the previous place to the right; i.e., the next less significant bit position. What is needed is a *full adder* circuit. To understand this, look at Fig. 10-28.

Assume you want to add two three-bit numbers, as in Fig. 10-27A. You proceed from right to left as in decimal addition, noting the carry bits as you go. At any given bit position, you must account for any bits carried in from the prior position before you record the sum at the current position; if there are any CO bits, you carry them over to the next more significant position to the left. Note that in the most significant position of this three-bit addition, there is a CI bit plus two one bits and a CO bit. This

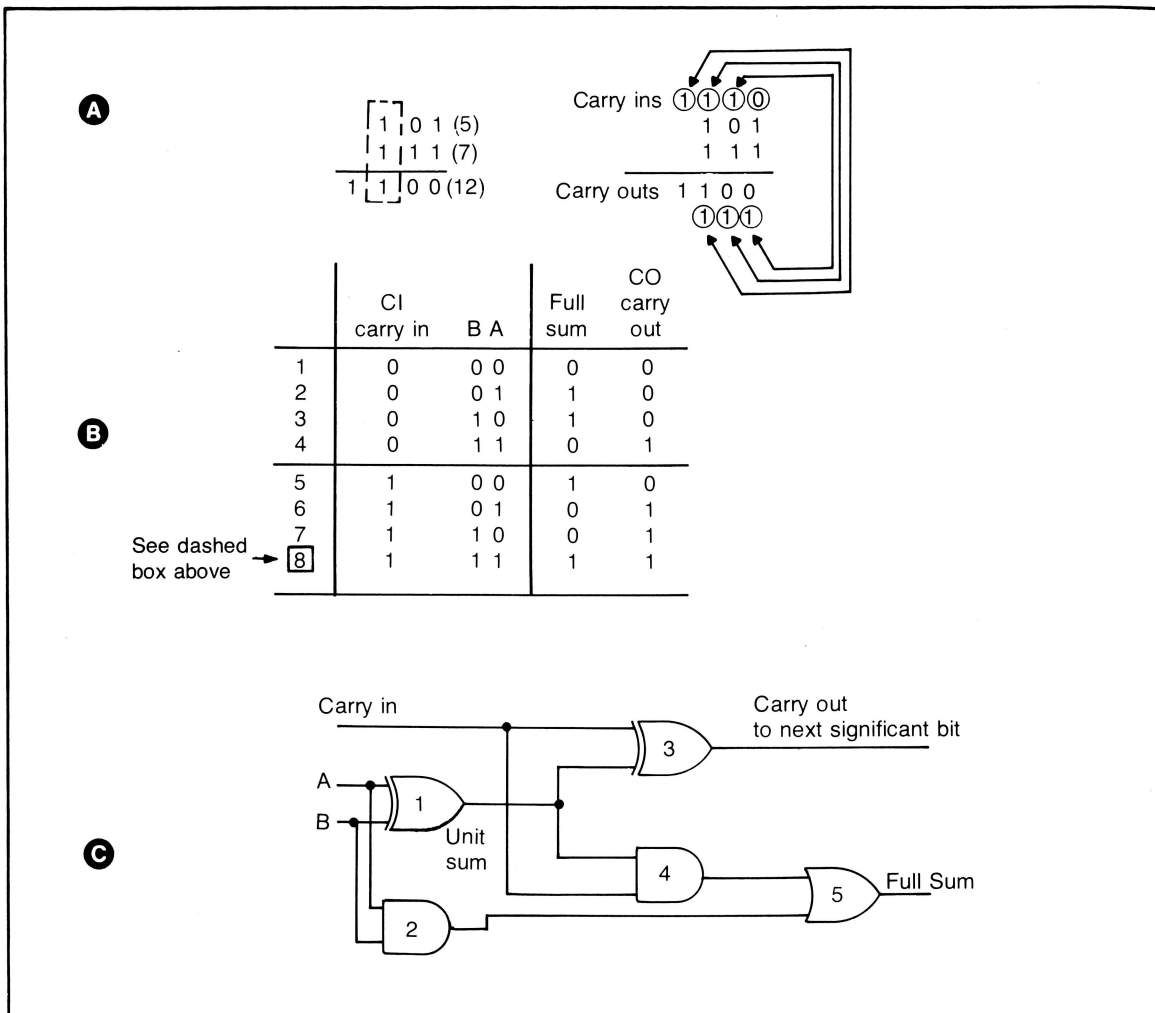


Fig. 10-28. The full-adder accounts for CI bits.

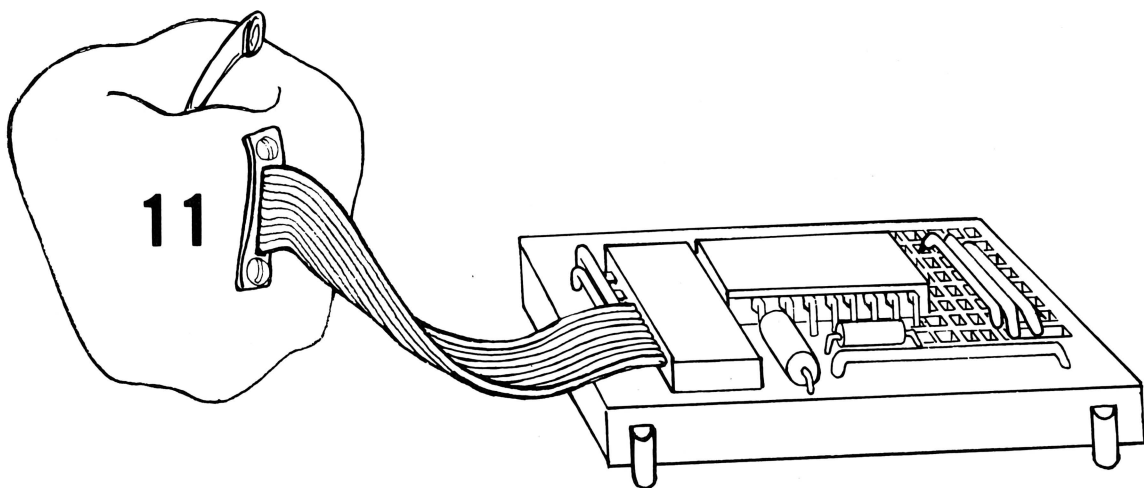
is indicated by the dashed box.

The truth table for all input conditions—status of A and B bits and of the CI bit—is given in Fig. 10-27B. The full sum at any bit position plus the CO bit constitute the outputs. This table describes a full adder and allows you to design the full adder circuit. (Line 8 corresponds to the dashed box in Fig. 10-28A). The circuit implementation is shown in Fig. 10-28C. Note that the CI bit of this circuit element would be connected to the CO output from the prior (less significant) element.

Naturally, four and eight bit adders come in

integrated form. Some of these are based on XOR logic. Many of these adders also perform other functions like subtraction, Boolean operations, and shift register functions. As you would expect, these *accumulators* and *arithmetic and logic units* (ALU's) fall into the upper MSI to LSI range of complexity.

Using your construction skills, you should be able to verify the operation of one or more of the XOR functions discussed. You might want to refer to the LS181, LS280, the LS281, and the LS283 in your data manual.



An Introduction to Interfacing

There is no question that microcomputer interfacing is a very broad and highly technical area that requires the proverbial library of volumes to do it justice. So, it may seem ironic that this final chapter is an introduction to a rather complex topic. Perhaps the subject should be reserved for another book. However, it really is appropriate to introduce you to the basics of interfacing at this time.

Microcomputer interfacing is the business end of your digital hardware knowledge—how you connect your computer to the outside world to make it do something useful, or at least interesting and educational. It is important that you get some idea of what is involved in interfacing right now. You should have no difficulty in this, because the principles of interfacing are only a logical extension of the material on digital electronics already presented. In this sense, this last chapter not only reinforces what you already know, but also helps prepare you for more advanced material that you will encounter later.

Early on in this chapter, we'll concentrate on

the major elements of computer hardware application that is referred to here as the I/O (input/output) *triad*. Once the broad picture is painted, we'll get into the specifics of the Apple's I/O memory organization and also illustrate the important topic of address decoding by direct reference to Apple's on-board I/O circuitry. In the concluding portion of the chapter, a few examples of applications circuits are presented. The emphasis is on measurement using resistive transducers. Each experiment is relatively simple, but this is intentional, as the simplicity permits a more detailed coverage of key elements in each project.

THE I/O TRIAD

The concern here is with computer-based hardware applications. To be clear on this point, we're not concerned with word processing, data management, spreadsheets, and the like, however powerful such programs may be. Rather, we'll deal with the other domain of computer applications—those tasks which cannot be performed by software,

no matter how clever it might be designed. For instance, you can't literally make a disk drive, a keyboard, a video controller, or an A/D (analog to digital) converter out of program code. That is, while software is a necessary part of any hardware configuration, you must also have the actual physical device and the all-important connection or interface between it and the computer it serves.

Thus, in any computer hardware application there are three main components: the *system* (computer and software), the *peripheral device* or applications circuit, and the *interface circuitry* which connects the peripheral to the computer.

In this scheme, the *system* is the brains behind the *peripheral*, and the *interface* is the necessary conduit for the data and various control signals between them. While the exact line of demarcation between any two of these components may sometimes appear blurred, this idea of computer hardware configurations as three-part entities is nonetheless fairly accurate.

There are an endless number of applications for computer-based systems. And as you might imagine, there are usually several approaches to a particular applications project—be it something basic like a printer card or more complex like a computer based instrumentation system. The final hardware/software configuration of the system-interface-peripheral may vary considerably. The configuration depends on the precise goals of the application, as well as on the skill and experience of the designer.

For example, there are many ways that you could design a modem communications device. The final solution depends on specific details of the application, and on whether it is to be cheap and simple, or may incorporate a number of sophisticated features.

However, in each such project, there are certain absolute *givens*. These are the things that the designer must deal with in almost any computer-based hardware project, particularly when he designs the interface between the computer and peripheral device.

In other words, there are certain tasks that the interface must perform regardless of the specific

nature of the I/O project at hand. Signal buffering, address decoding, data latching, and synchronization between computer and peripheral are always required. **In essence, the interface is a slave that must serve two different masters: the system and the peripheral.**

Let's take a look at the different demands that may be made on the interface, and on the functions which it must fulfill.

System Demands

The microcomputer system itself is a given, and the interface must serve this element of the I/O triad. More specifically, you are provided with a bus oriented system with data, address and control lines. And it is this raw system bus which must be interfaced with your applications circuit.

System Bus. The system bus refers to the data, address and control buses, a concept you've encountered earlier.

The data bus is bidirectional, as data may originate in either the microprocessor or in the peripheral. It has eight parallel lines, as the typical data word is eight bits wide. (An extra parity bit is present in some micros as well).

The address bus is unidirectional, because in most situations the computer is addressing the peripheral, not the other way around. (The exception is so-called direct memory access (DMA) in which the peripheral takes control of the system address bus and accesses computer memory directly, independent of the microprocessor. This is an advanced subject of which we'll say no more). The address bus typically consists of 16 address lines, corresponding to the 16-bit address word in an 8-bit micro. A total of 65,535 memory locations can be accessed with such a bus. Longer address words in 16-bit micros exist.

Last, there is the control bus, which is a mixed collection of lines: read/write output lines from the microprocessor that control the direction of data flow; interrupts, which are input lines to the microprocessor; the system clock lines, which help synchronize microprocessor operations.

The main point here is that these bus lines

have certain electrical characteristics that must be respected by any circuitry connected to them. The voltage and current specifications cannot be exceeded. In the case of the Apple's 6502 TTL microprocessor, it is obvious that TTL guidelines must be heeded.

In short, the interface must assure that signals into and out of the computer are electrically compatible with the TTL requirements of the computer's system buses.

Memory Mapping. Beyond these physical considerations of the system bus, there is another system "given" which is of a more abstract nature: the computer's I/O memory organization.

The Apple treats peripherals attached to its system bus just as it treats main system memory. That is, as far as Apple is concerned, the keyboard, the disk drive or the printer are nothing more than memory locations. There are no special, dedicated I/O lines coming off the 6502 microprocessor chip. There are only address lines that select the desired location in memory—either writing data to that location and/or reading data from it. Whether that location in memory is actually a byte of RAM or an external device makes no difference to Apple's microprocessor. This form of memory organization where normal RAM/ROM access is not segregated from I/O functions is known as *memory-mapped* I/O.

Thus, in the 6502 type memory-mapped I/O system, the interface must perform address decoding by using address bus logic levels to access the peripheral device.

Some microprocessor families do in fact have dedicated I/O lines. Such chips are said to have *I/O mapped* input and output organization. The 80xx series of microprocessor chips is a prime example. Naturally, there are proponents of both systems. However, the chief advantage of memory mapped I/O organization like Apple's is that you have potentially as many I/O ports as you have locations in memory. This is particularly useful in those settings (factory automation, security, scientific data collection) where hundreds of points might have to be monitored or controlled by the computer. Also,

from the programmer's viewpoint, memory-mapped I/O is more convenient because no special instructions are needed to write to or read from the peripheral device.

The Software. The prime function of the interface is to transfer properly formatted and electrically compatible data between the computer and the peripheral. Once this is done, it is the role of the software to store, process and display this data. You need to be able to POKE and PEEK data in BASIC, or do the equivalent from assembly language. You must also be able to store the data for subsequent processing; this storage may be in the form of simple integer variables, as strings of text characters, or as a multidimensional floating point array. Finally, you must be able to display the data going to or coming from the peripheral in a readable form—single number display or alphanumerics, tabular display, running text, elaborate graphics or whatever.

The software component of the system, then, makes no demands on the interface. Instead, it really is a servant of the whole I/O triad.

As usual, there are some qualifications. Occasionally, the optimum dividing line between hardware interface functions and software functions is not always as clear as implied above. The designer/programmer must decide what tasks his project are best handled by hardware, and what tasks are best performed by software. This is not always easy, but once the circuit functions have been partitioned from the code, the hardware design becomes clearer.

The decision must also be made as to how the software that controls the peripheral circuit is to be embodied. Storing the program on disk or tape is a possibility. In the many cases, however, the peripheral and its software should ideally be part of the same package, not separate. Usually, the most efficient way to do this is to *burn* the program (most often in machine language) into a PROM (programmable read only memory). This PROM is then plugged into and becomes part of the peripheral circuitry. Being neither pure hardware (it's intelligent) nor pure software (unlike tape or disk), the

PROM is in between. It's therefore called *firmware*, a term you've probably heard before.

In summary, the interface design must take into account the following:

- A raw system bus, with certain electrical characteristics, often TTL.
- Memory-mapped I/O organization, in which address decoding is necessary to activate the peripheral.
- There is necessity for some resident software for your I/O application. Therefore, consideration given to the optimum partitioning of hardware tasks from software tasks.

Demands of the Peripheral Device

This is the functional part of the I/O triad. Examples include instruments for measurement, magnetic storage devices (disks), information displays (monitors and printers), data input devices (keyboards, graphics tablets), modems, servomechanisms (robots!), and so on.

Obviously, it is almost impossible to specify a list of fixed demands that a peripheral will impose on an interface. This is due simply to the seemingly countless list of applications circuits that exist. However, it is possible to mention some of the factors that should be considered by the designer when he approaches such projects. A partial list of factors involved in interface design relative to the peripheral would include the following:

Data Format. This is an important factor in interface design. That is, will data be in serial or parallel format? Will data be transferred as a string of bits, or in parallel 8 bit data words or bytes. If in serial format, are there other special constraints, such as communications protocols? Modems, networks, and multiuser applications often involve long distance communications; therefore, some form of serial data protocol is mandatory. Laboratory instruments, on the other hand, are often quite close to the desktop microcomputer. Therefore, the parallel data format is often used, especially since the signal lines are more easily mated to the

parallel system data bus.

Signal Conditioning. This set of demands refers to the electrical characteristics of the peripheral's signals. We've already said that signals to and from the system bus must adhere to TTL specifications: square waves with clean rising and falling edges, proper voltage levels, safe current levels, and as little noise as possible. Here we're referring to the opposite side of the problem; the interface signals must also be compatible with the peripheral's electrical needs.

For example, there are circuits which require lots of power at the input, deliver a lot of current at the output, or which control other power hungry devices (motors, incandescent lamps, etc.). Further, these peripherals may generate or require data/control signals of odd (non-TTL) voltage levels. In some circuits, further electrical isolation via relays or opto-isolators may be necessary. The interface must provide for these electrical requirements in some way or another.

Dynamic Factors. Timing, speed of operation, and "handshaking" are other factors which may enter into the design of an I/O system, depending on the nature of the peripheral.

Timing implies synchronization of computer with peripheral. For example, a small but finite time is required for the computer to set up the address lines before it transfers data onto or from address bus. The easiest way of assuring the proper timing of these operations is to use the system clock signal. This signal may control part of the interface circuitry, so that the transmission or receipt of data occurs at the proper time.

Speed of operation is also a factor. High speed operations may require not only parallel data transfer (as opposed to serial), but also may require *hardware intensive* implementations. For instance, suppose that you want to display quickly changing data in real time—say in graphic form as the measurements are being made. In such cases, hardware processing of the data may be necessary, as this is usually much faster than software data processing. A prime example of such an application is in high resolution medical imaging. Pictorial data from a

transducer is digitized, cleaned of noise, formatted and finally sent to video memory for display at high frame rates. Even machine language is too slow to perform all of these tasks on the raw data. So some hardware preprocessing is mandatory if a high quality, flicker-free image is to be obtained.

Handshaking is another function which may be incorporated in an applications circuit. Handshaking is best defined as a form of communication between computer and external device; it is most often employed when the speed of operation of the two differs widely. Handshaking accomplishes synchronization between computer and peripheral much as do common clock signals, though in a different manner.

For instance, a modem operates much more slowly than the computer which sends it data for serial transmission over the phone line. The same thing may be said of a printer. In either case, the peripheral will send a *ready* signal to the computer, telling it when to send the next byte (or block) of data. Once the data is received, it may send an *acknowledge* signal. After doing what it has to do (printing, transmitting, etc.) the peripheral will then send the next ready signal. This handshaking, or exchange of ready and acknowledge signals between computer and peripheral, is an asynchronous process because the system clock signal does not completely control data transfer.

Analog or Digital Data. This is the another basic question you would ask about the external device. Are the signals originating from or destined for the device digital (e.g., on/off switches) or analog (as from a temperature probe)? If the signals are already in digital form, then so much the better.

For instance, simple TTL devices and circuits are no problem. Witness the BDIS/game port-based logic trainer system; this is after all nothing but a digital-to-digital system.

But if the external device is generating analog signals (or requires such signals), then some form of analog-to-digital (A/D) or digital-to-analog (D/A) converter is necessary. An example of a need for A/D conversion would be that of measurement; if the computer was to output sound or voice, then

D/A conversion would be required.

Summary of Interface Functions

To get back to the essentials, we can reiterate the functions that all interface circuitry must fulfill: At the very least, the interface must provide the system (computer + software) access to a specific peripheral device and assure that the data and control signals are electrically compatible with both the system bus and with the peripheral device. The first function is that of address decoding; the second implies, at the very least, buffering and latching of signals.

Address Decoding. Since in memory-mapped I/O systems each peripheral occupies at least one location in memory, the appropriate lines in the system address bus must be used to activate or address the peripheral. Remember how the LS138 1-of-8-line decoder operates: one of eight lines goes low for a given 3-bit address on the data-select lines. These lines in turn may be used to selectively enable another device or circuit.

In other words, the whole purpose of address decoding circuitry is to translate voltage levels on the system address lines into an active logic level called a *device select* or *chip select* signal which will serve to enable and disable the circuit or device of interest. Then, data and control signals can be freely transferred between the system and peripheral.

Data and Control Signal Transfer. The other major task of the interface circuitry is to assure that data and control signals are efficiently exchanged between the computer and the interface. The primary functions would include temporary data *latching* or storage in registers, *buffering* the signal to the proper strength (fan out), and providing for bus *isolation* if necessary through the use of three-state logic.

Control signals also may play a part in the operation of the interface and the applications circuitry. The microprocessor's read/write line must be employed if there is to be bidirectional data transfer. System clock lines (there being more than one in most microcomputer systems), as well as

Table 11-1. The Main Elements in the I/O Triad.

System aspects	Raw system bus—TTL electrical specifications of the data, address and control buses. Memory mapped I/O—external devices treated as locations in memory. Applications software—RAM - machine or high level, language or a mix of the two. ROM/PROM - machine code in firmware.
Interface aspects	Address decoding circuitry—selective enabling/disabling of the peripheral device by use of the system address lines. Signal transfer—routine handling of system bus signals, e.g., buffering, latching setting data direction, etc.
Peripheral Device (factors in interface design)	Analog or digital data Data format (serial or parallel) Signal conditioning (proper voltage, current levels) Dynamic factors (timing, speed, need for handshaking)

interrupt, reset, DMA and other lines in the *control bus* may be required, depending on the application.

Non-TTL voltage levels may be generated or required by the peripheral, so that conversion to TTL may be another task of the interface. The same thing holds for data format and for analog/digital interconversion.

The division of applications hardware into system, interface, and peripheral is perhaps an oversimplification, as it may be argued that the distinction between these elements is not so sharp as suggested here. It may be pointed out by some that A/D and D/A conversion functions are properly classified as functions of the peripheral itself, and this is certainly reasonable.

Admittedly, the subjects of interfacing and I/O applications fall into the intermediate to advanced range of technical expertise, not only in terms of hardware design, but also in terms of the design of appropriate software. Hopefully though you at least have some familiarity with the major factors involved in interfacing. A summary of the main ideas so far presented is given in Table 11-1.

THE SYSTEM AND I/O MEMORY MAPS

To this brief overview, we will now add or plug in a detailed example of an I/O triad—the Apple's I/O memory organization, address decoding concepts and on-board decoding and I/O circuitry, and a few simple applications examples using resistive transducers. The important subject of address decoding will become a lot clearer as we analyze the particulars involved. In this way you will have a much better conception of the subject matter. Some mention of slot-based I/O will also be made during the discussion.

Computer Memory, Blocks and Pages

A computer's three basic operations are to input, process, and store data. Memory is required for input, for storage, and for the display of the results. And though it is the microprocessor chip that performs the actual operations on the data, it is memory which holds the instructions necessary to perform these operations.

Computer memory consists of both RAM (random access or read/write memory) and ROM (read

only memory). RAM memory is programmable memory which can be changed at will, either by programming from the keyboard, or loading from external storage devices such as floppy disk drive. When the computer is turned off, the data contained in this *volatile* RAM is lost. ROM memory, on the other hand, is permanent. It contains fixed programs which are necessary to the operation of the particular machine—such things as primitive routines that are used for routine input and display of data, as well as more sophisticated programs such as high level language interpreters. The Apple's monitor ROM and Applesoft ROM are examples of such hard-wired programs. This type of memory is known as *firmware*.

As mentioned in Chapter One, in a typical 8-bit microcomputer the data word is 8-bits wide (one byte) and the address word is 16-bits wide; this corresponds to the 8 and 16 lines on the data and address buses respectively. The number of distinct and separate memory locations available in an 8-bit computer would be 2^{16} or 65,536. Each location holds an 8-bit data word. This is expressed as 64 kilobytes of capacity, or 64 K for short.

A 64 K memory range can be expressed by a four digit hex number: \$ZYXW, each digit assuming a value between \$0 and \$F (0 to 15 decimal). Each hex digit, from the least to most significant digit (digit W to digit Z), has its own place value, just like in the decimal number systems. In this case, the place values are 1, 16, 256, and 4096. (See the section on number systems in Chapter One). The entire 64 K range can be expressed, then, as \$0000 to \$FFFF.

Any given hex number—\$D1F5 (decimal 53749) for instance—represents more than a simple numerical value. It also represents a location in memory, a place where data is stored. This 4 digit hex number is a memory address, which can be accessed using the appropriate decoding hardware.

Related to this idea of memory range is the concept of *memory blocks* and *pages*. A block of memory is basically any sub-range of memory. In a 64 K system one might speak of 2 K, 4 K or 16 K blocks. Relating this to hex notation, you can see that a 4 K block is represented by the most significant

digit in the 4-digit address: The most significant digit has a place value of 4096, that is, there are 4096 locations in the range \$C000-\$CFFF, or in the range \$1000-\$1FFF. An example of a 16 K block of memory would be \$C000-\$FFFF, and of a 2 K block, \$C000-\$C7FF.

In a more restricted sense one can speak of a page of memory. This has a specific meaning: a 256 byte block of memory specified by the two most significant hex digits in a 4-digit hex number. For instance:

\$0000-\$00FF page zero (\$00) and
\$A300-\$A3FF page 163 (\$A3)

are two pages of memory. But \$0080-\$017F is not.

The reason why the third range is not technically speaking a page—even though it contains 256 bytes—is that it crosses a *page boundary*, that is from page zero to page one. The first two ranges can be expressed by two hex digits (\$00 and \$A3). But because the third range crosses the page boundary, it must be specified by a 4-digit range. Hence it does not belong exclusively to one page.

These concepts of pages and blocks are useful when discussing memory mapping, the next subject.

System Memory Map

The allocation of this computer memory is

SYSTEM ROM FIRMWARE (Monitor and Applesoft)	12 K	\$FFFF \$D000
I/O MEMORY	4 K	\$CFFF \$C000
FREE RAM MEMORY	46 K	\$BFFF \$0800
RAM RESERVED FOR SYSTEM USAGE	2 K	\$07FF \$0000

Fig. 11-1. Rough system memory map.

depicted by means of a system memory map. As you would expect, (and we'll restrict the discussion to 8-bit computers) each type or family of microprocessors uses its 64 K differently. But all have ROM for system instructions (monitor, built in BASIC, etc.) and RAM for CRT display, operating system software, application programs, temporary data storage, etc.

Figure 11-1 illustrates the system memory map for a typical 8-bit machine, the Apple. Shown in this simplified map is the 64 K memory range from \$0000 to \$FFFF, with the highest location (65535 decimal) at the top.

The emphasis here is on the use of the major ranges of memory, of which there are four types: space for a high-level language and monitor in ROM, an area for input and output (I/O) functions, space for application programs in RAM, and temporary storage areas for system usage, also RAM memory.

Keeping in mind these four ranges of memory, we can delve into a more detailed description of system memory organization, given in tabular form this time, in Table 11-2. To keep things simple, assume we have a so-called 48 K machine with DOS.

From top to bottom, the four areas of memory can be defined in terms of both size and function. In regard to size, we can refer the number of individual bytes, or refer to the range in terms of page numbers, expressed as hex or decimal. This is done to the left in Table 11-2. As to the functions of each major range and sub-range of memory, we can break them as follows:

1. System Firmware (pages 208-255). In the Apple, 12 K at the upper end is occupied by the burned-in software (PROM or ROM) that comes with the machine. This consists of the Applesoft BASIC interpreter and a 2 K monitor program to take care of housekeeping chores like scanning the keyboard, outputting text to screen, plotting in LORES graphics, etc.

For those of you who are not familiar with 16 K memory cards or with Apple IIe banked memory, it should be mentioned that this 12 K upper block can

serve another purpose: this same 12 K block can alternatively be occupied by 16 K of RAM memory, in a 12 K + 4 K arrangement! This RAM must be bank selected by soft switches dedicated to that purpose. Again, soft switches are memory location which when accessed can effect a change in hardware function, in this case activation of alternate memory banks or blocks. This is done both in the Apple IIe, and in those Apple IIs, with 16K RAM cards. In effect this rather odd arrangement places two blocks of memory in parallel.

2. I/O Memory (pages 192-207). A 4 K block on the Apple in the \$C000-\$CFFF range is devoted to input/output. Most of this block is reserved for use by the peripheral cards that occupy the I/O slots on the Apple main board. Half of it, 2 K or eight pages, is intended for PROM memory space that is shared by the cards. The other eight pages is divided among the I/O slots and for on-board I/O functions. These are detailed shortly.

3. RAM for applications programs (pages 8-191). Nominally, 46 K (more precisely 47 K) is set aside as programming space, that is, space where both the program currently in use resides; data generated by the program is also stored in this free RAM area.

This area is not entirely free, however. In reality, a fair amount of this space is consumed by a special applications program Apple's DOS (Disk Operating System). DOS takes up roughly the upper 11 K.

There is also a 16 K area which may, if desired, be used for two pages of HIRES graphics display; also, there is another 1 K at the bottom of this range that may be used for a second page of text of LORES graphics. If these latter options for HIRES and secondary text/LORES memory usage are not required, then about 36 K is available to the user for program and data space. (Check the DOS and A/S Manuals for further details about memory usage).

4. RAM for System Use (pages 0-7). About 2 K is reserved for usage by the A/S interpreter, the Apple Monitor and the DOS. This space includes

Table 11-2. Breakdown of Apple's System Memory Map.

64 K or \$FFFF bytes of memory
is organized as
256 (\$FF) Pages of 256 (\$FF) Bytes each.

Page number Decimal Hex	Function or Allocation	Decimal Address Range
255 through 208	System monitor and Applesoft ROM	65,535 to 53,248
207 through 192	Apple's I/O Memory	53,247 to 49,152
191 through 96	Disk Operating System (10.7 K)	49,151 to 24,576
95 through 64	Free RAM	24,575 to 16,384
63 through 32	Free RAM	16,383 to 8192
31 through 12	Free RAM	8191 to 3072
8-11	Free RAM	2048-3071
	TEXT and LORES Display [Secondary Page]. \$800 is usually the beginning of A/S programs.	
4-7	TEXT and LORES Display [Primary Page]	1024-2047
3	Page Three - Mostly free, some vector locations	768-1023
2	Page Two - Keyboard Input Buffer	512-767
1	Page One - System Stack	256-511
0	Page Zero - Monitor/DOS/Applesoft Usage	0-255

about 1 K of video memory for primary page text display, and temporary storage locations for DOS, monitor and Applesoft. (Pascal, COBOL, Logo and other high-level languages also use some of these locations). You may be aware that there is space on page 3 and a few free locations in page zero for assembly language programmers to squeeze in short routines. Otherwise, this lower 2 K block must serve varied system functions and, with these few exceptions, should not be written over by the programmer.

The I/O Memory Map

From here on in we'll concentrate on the I/O block of memory. This will allow you, ultimately, to understand memory mapping and related concepts right down at the chip level!

Figure 11-2 is a blowup of the 4 K (16-page) \$C000-\$CFFF memory block that, on the Apple, is dedicated to I/O functions. The first major division of this block is into slot-based I/O functions and on-board I/O functions. The slot-based range occupies \$C080 through \$CFFF; the on/board I/O range occupies \$C000 through \$C07F. Using Fig. 11-2, and the more detailed breakdown given in

Table 11-3, we'll discuss the various functions in these two ranges.

On-board I/O Memory. (\$C0nx n=0-7). At the bottom of the 4 K I/O memory block is a 128 byte (½ page) from \$C000 to \$C07F. As you can see from Table 11-3, this set of locations is devoted to functions performed ON-BOARD, that is, as an integral part of computer operation. These are exemplified by such functions as holding a character sent from the keyboard, toggling the speaker, setting screen display soft switches, and inputting and outputting signals to and from the game connector. The locations we are concerned with in our computer based logic trained (BDIS) are marked by asterisks.

Note especially the hex notation for a ½ page memory range. The page \$C0 might be represented by the notation \$C0xx, where each 'x' can assume any hex value from 0 to F. But we want only to specify one-half of this page, so the notation \$C0nx is used, where n=0-7. This notation indicates the 128 byte range from \$C000 to \$C07F which on-board I/O occupies.

Slot-based I/O Memory (\$C080-\$C0FF). Slot-based I/O refers to memory space allotted to

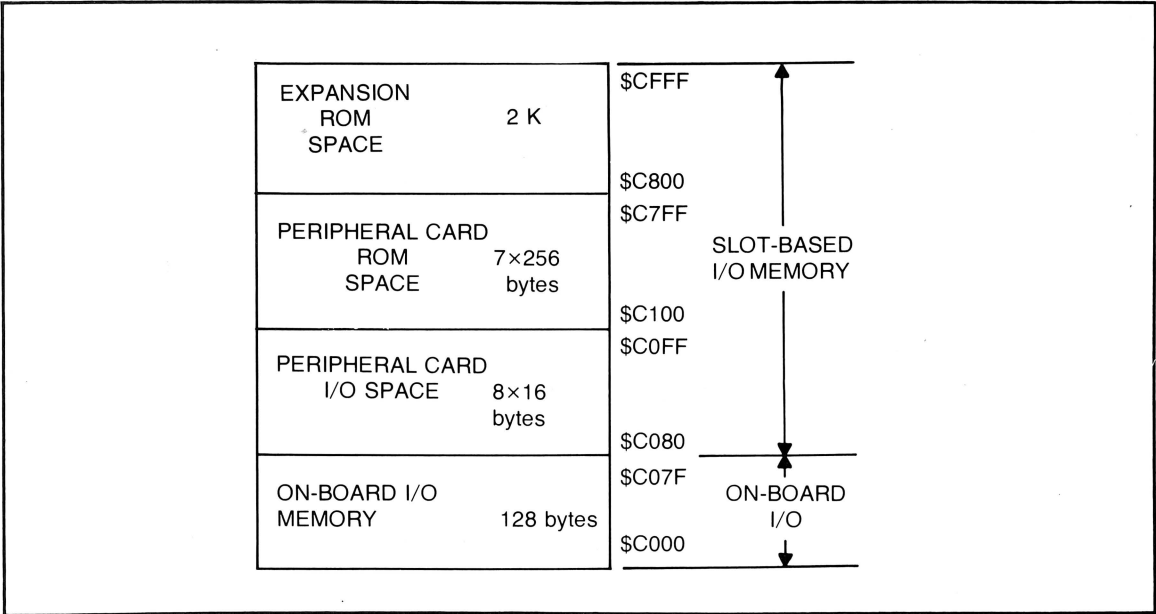


Fig. 11-2. I/O memory map.

Table 11-3. Detail of the I/O Memory Allocations.

Hex Address or Address range	Function
Expansion ROM: \$CNxx N=8–F 2 K Shared	
\$C800 - \$CFFF	This is the upper 2 K of I/O memory which is shared among the peripheral I/O slots. A 2 K PROM on a given card can be switched into memory, using address decoding signals I/O select and I/O strobe.
Peripheral Card ROM Space: \$CNxx N= 1-7 7×256 bytes	
\$C700 - \$C7FF	Seven pages of 256 bytes each are set aside, one page for each peripheral card in slots 1 to 7. Slot 0 has no such reserved page. In the notation \$CNxx, N is the slot number.
through	
\$C100 - \$C1FF	
Peripheral Card I/O Space: \$C0nx n=8-F 8×16 bytes	
\$C0F0 - \$C0FF	Eight little 16 byte blocks are set aside, one for each peripheral card in slots 0 to 7. The exception is the] [e, in which \$C080-\$C08F is devoted to memory management. In the notation \$C0nx, n=slot# + 8.
through	
\$C080 - \$C08F	
On-board I/O locations: \$C0nn n= 0–7 128 bytes	
\$C000	Keyboard Data and Keyboard Strobe Bit
\$C010	Clears Keyboard Strobe Bit
\$C020	Cassette Out
\$C030	Toggle Speaker
* \$C040	Utility Strobe
\$C050 - \$C057	Screen Soft Switches (Page/Text/Graphics)
* \$C058 - \$C05F	Annunciators (4 pair of ON/OFF switches)
* \$C060	Cassette input
* \$C061 - \$C067	PB0-2 Inputs, and PDL0-3 Inputs
* \$C070	Strobe for PDL inputs (triggers 558 one-shot)

Note: Range \$C068-C06F repeats the functions of \$C060-C067. Also, the] [e uses some locations in the \$C00x and \$C01x ranges for memory and display functions.

the eight I/O peripheral card slots on the back of Apple's main board. These 50-pin connectors carry the full address bus, data bus and control bus, along with four different voltages (+/- 5 and +/- 12 volts) and a few extra address decoding signals.

The slot-based memory is broken down into three other subranges:

Peripheral card I/O space (\$C0nx n= 8-F). The other half of page \$C0 is taken up by 128

bytes devoted to the peripheral cards. The organization is 16 bytes for 8 slots (numbered 0 to 7), for a total of $8 \times 16 = 128$. These locations are often devoted to “configuring” the card for a specific application rather than for program storage. Note that slot 0s 16 bytes are given over to other functions in the *IIf*, as noted in the table.

Peripheral card ROM (PROM) space (\$CNxx N=1–7). Seven pages are set aside for slots 1 to 7. Here, the capital N signifies a full page for each card. Each 256 byte page can be used for a little machine language code to run the peripheral card in the given slot. Sometimes, such a short program burned into a 256 byte PROM is sufficient. For more sophisticated functions—printer cards with dot addressable graphics for example—more memory is required, so-called expansion ROM.

Expansion ROM (PROM) space (\$CNxx N=8–F). Last, there is a 2 K block of expansion ROM. This space is shared by all slots and is usually occupied by 2 K worth of firmware which is on the card. Modem cards, printer cards, and the like therefore have 2 K worth of program memory space available to them, space occupied by their built in PROMs. Since only one peripheral card can be active at a time, the respective 2 K of PROM on the active card (containing the card’s driving software) will “take over” this upper 2 K I/O memory block. The other cards, and their 2 K PROMs, are inactive, so that no conflict occurs. (The scheme for this is given on pages 84-85 of the Apple II Reference Manual, and on pages 123-125 of the Apple *IIf* Reference Manual).

Don’t hesitate to read and reread the above material on memory maps, with Reference Manuals in hand, because this concept of memory organization is necessary in understanding the next section.

I/O ADDRESS DECODING

Some of the other tasks the interface must perform—buffering, latching, isolation via three-state logic—have already been covered. Therefore, we’ll deal here with the other major task of interface circuitry, address decoding. The I/O block will be used as the example, however, the

principles are the same regardless of the memory range involved.

Remember that a peripheral device occupies one or more places in memory in memory mapped I/O systems like the Apple, so that device selection involves decoding the address lines attached to that device. Technically, anything external to the microprocessor chip is a “peripheral” including on-board RAM and ROM. This is reasonable because all addresses, whether associated with RAM memory or a robot arm, look the same to the Apple’s 6502 microprocessor. Therefore, we can see how address decoding is accomplished simply by examining the decoding circuitry on the Apple’s main board.

First we’ll look how large blocks of memory are opened up by upstream decoder chips. Then we’ll see how smaller blocks and pages, and finally individual locations are addressed.

Block Decoding

Address decoding begins with the activation of the proper address lines in the 16-line address bus. The address lines run from computer to the address decoding circuit, and the output of the decoding circuitry consists of one or more lines which will in turn activate one or more devices.

With 65,536 locations in an 8 bit microcomputer (there are never any more than this at one time, even with bank switched RAM memory, RAM disks, and the like) you may ask if the decoding is accomplished all at once. This is not the case because, practically speaking, the circuitry would be too complex. Instead, address decoding is achieved stepwise, by selecting a large block and then proceeding to small blocks, pages, sub-pages and finally to individual locations in memory.

Figure 11-3 shows how 4 K blocks of memory can be opened up in a 64 K system. By using the top four address lines A12-A15, one can decode for one of sixteen 4 K blocks, for a total of 64 K. This is accomplished using the LS154 1-of-16 decoder. Address lines A0-A11 are employed for downstream decoding circuitry, which will select for smaller and smaller ranges of memory.

Note that our range of interest—I/O Memory Range \$Cxxx (\$C000-\$CFFF)—would correspond to output line 12 or hex \$C. This line goes active low when binary 1100 / hex \$C is placed on the input. Figure 11-3B shows the status of the address lines, with 'x' indicating "don't care", or in other words 0-F.

Is this the way Apple begins its stepwise decoding? Actually, no. The LS154 example was used to present the basic idea of block decoding in a simplified form. What really happens on the Apple main board is a bit more complicated, and is presented below.

I/O Decoding Circuitry

The details of the I/O decoding circuitry are given in this and the next section. This circuitry is also presented in the main board schematic in the Apple II Reference Manual. The *Ite* has consolidated many chips present on the II and II+ into custom ICs—the Memory Management and I/O Units. The details of address decoding are lost or buried in these large custom chips and so are not visible on the schematic in the *Ite* manual. Therefore, if you can, try to borrow or buy the original Apple II Manual. This manual has a fold-out schematic of the entire main board, so that you can see the on-board circuitry at a glance. In all our discussions, we will be referring to the original circuitry on the II/II+. Again, the principles are the same, whether the decoding circuitry is buried or not.

I/O address decoding is done in several states stages, in which the memory is broken down as follows:

- ☐ first from a large 16 K block to eight 2 K blocks using the ROM select chip,
- ☐ then from a 2 K block to eight 256 byte pages using the I/O select chip,
- ☐ then further into two 128 byte half-pages (device select and on-board I/O decoder chips),
- ☐ one of which is divided into eight 16-byte sections for I/O slot usage, the other being devoted to a variety of on-board I/O functions.

Now to cover each stage in turn.

ROM Select Chip (16 K - pages \$C0-\$FF).

What the Apple does first in decoding I/O memory is to use the LS138 1-of-8 decoder to select the upper 16 K of memory, and further break it down into eight 2 K blocks. This chip is called the ROM select chip on the schematic in the Apple II/II+ manual. Refer to Fig. 11-4A and B.

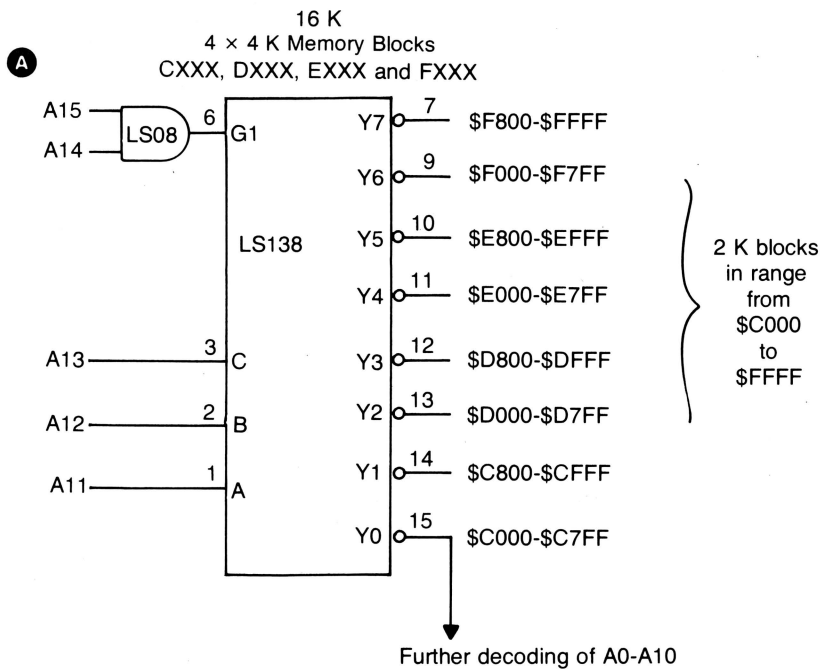
How is this selection of the upper 16 K accomplished? Simply by tying both A14 and A15 through an AND gate (LS08) to the active high enable input, G1. Only when this input is active (with both A14 and A15 high,) is the LS138 decoder enabled. When so enabled, the LS138 chip selects address locations in the range \$Nxxx where N=\$C to \$F. If either or both A14, A15 are low, then it means the microprocessor is addressing the lower 48 K of memory.

With the upper 16 K selected by A14 and A15, the chip then decodes for eight 2 K blocks in this 16 K range. The output line for each 2 K is active low. Note how input lines A11-A13 determine which output is selected.

Specifically, lines A12, A13 select which 4 K block is turned on: \$Cxxx, \$Dxxx, \$Exxx or \$Fxxx. Then line A11, the least significant bit in the 3-bit input, selects the upper or lower half of the 4 K block. In the case of the \$Cxxx block, when A11 is 0, \$C000-\$C7FF is selected; when A11 is 1, \$C800-\$CFFF is selected. This is illustrated in Fig. 11-4B. The same principle applies to the other 2 K ranges selected by this chip.

I/O Select Chip (\$CNxx N=0-7). Our interest is, ultimately, in seeing how the on-board locations (including the game port addresses) are decoded. So we'll look at the 2 K range off the ROM select from \$C000-\$C7FF, i.e., \$CNxx (N=0-7). This is the "Y0" output off the ROM SELECT. Using this line, and three more address lines A8-A10, we can decode for eight smaller ranges in this 2 K block. Again, the LS138 is employed, as shown in Fig. 11-5. This chip is known as the I/O select chip.

As you can see in the figure, the eight sub-ranges decoded are actually eight pages. The upper seven pages, \$C1 to \$C7, are used to select for 256



B

	A15	A14	A13	A12	A11	(A10-A8)	(A7-A4)	(A3-A0)	Range decoded
Y1 $\xrightarrow{\text{CNXX}}_{\text{N=8-F}}$	1	1	0	0	1	XXX	XXXX	XXXX	\$CFFF \$C800
Y0 $\xrightarrow{\text{CNXX}}_{\text{N=0-7}}$	1	1	0	0	0	XXX	XXXX	XXXX	\$C7FF \$C000

↑
For 128 bytes I/O space
and 128 bytes on-board I/O

Fig. 11-4. ROM select chip, with 2 K blocks. Breakdown of block \$C is shown in (B).

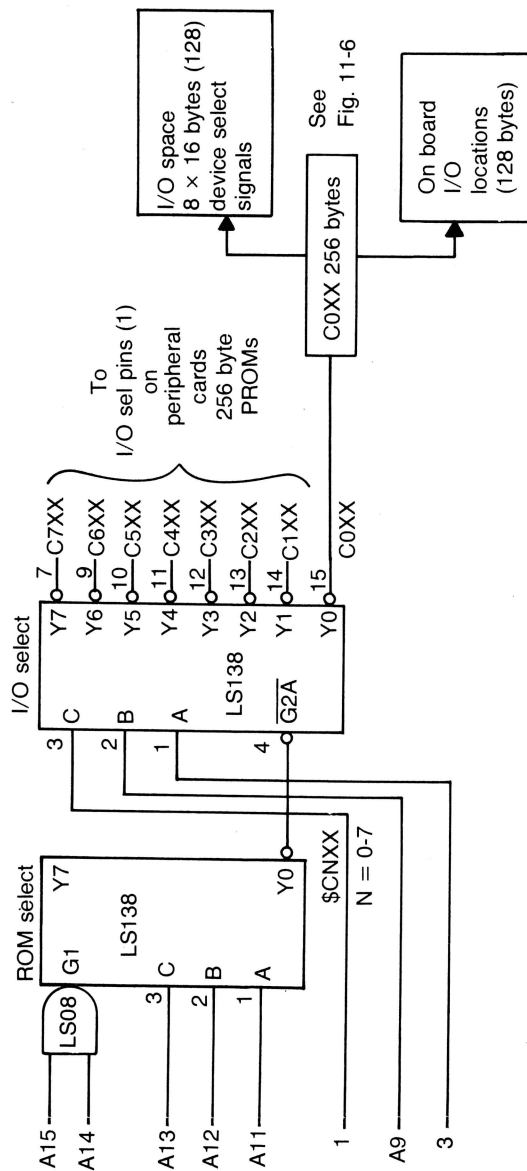


Fig. 11-5. I/O select chip, generating I/O SEL signals. Further decoding is performed on page \$C0, as shown.

bytes of PROM/ROM, which contain short machine language drivers on the peripheral cards that may reside in these respective slots. These are the peripheral card ROM space pages mentioned earlier. The signals coming off outputs Y1 to Y7 are called the I/O select signals, and are active low. They are distributed to pin #41 on each of the slots 1 to 7.

Observe that peripheral slot 0 has no such page set aside for it. Instead, page \$C0 is broken up into two 128 byte half-pages, one for I/O space, the other for on-board memory.

Device Select Chip (for I/O space) (\$C0nx n=8–F). Refer to Fig. 11-6, which recapitulates the address decoding circuitry we've covered so far.

The 128 bytes from \$C080 to \$C0FF are decoded into eight groups of 16 bytes each. These are available to each of the eight I/O peripheral slots. This breakdown of the upper half of page \$C0 is done by another LS138, called the device select chip. Note that it is activated by the \$C0xx line off the I/O select chip.

Further decoding is done by the four address lines, A4 to A7. When A7 is high, the device is active for binary inputs of 1000 to 1111. (It's connected to active high G1). This corresponds to the N=8–F range of \$C0nx. Each value of 'n' covers a subrange of 16 bytes.

Often, these locations (16 to a card) are used to set up or configure the card, such as by turning flip-flops on the board on or off, for example. They often function, then, as soft switches, rather than as space for program storage.

On-Board I/O Chip (\$C0nx n=0–7). Another LS138 chip is used to decode this 128 bytes of page \$C0, (lower half-page) into eight 16 byte groups. The only difference between the address line connections to this chip and that of the device select chip is in line A7. This line is connected to an active low enable, G2A', instead of an active high enable. Therefore, chip will be operational as a decoder when the inputs are in the range of 0000 to 1111. This corresponds to the n=0–7 range in \$C0nx. Again, each output line corresponds to a 16 byte group. Major functions of each group are indi-

cated in Fig. 11-6.

Further decoding of the on-board signals is covered below.

ON-BOARD I/O CIRCUITRY

The eight groups of 16 bytes each that fall into the lower half of page \$C0 are the on-board memory I/O locations, just covered in the last section. As you can see from Fig. 11-6 and Table 11-3, the locations in this 128 byte range are either inputs or outputs.

Outputs

The major output lines with which we're concerned are the four annunciators (AN0 to AN3), and the utility strobe (UTILSTB).

The UTILSTB line is taken directly off the on-board decoder, as indicated in Fig. 11-7. Note that the role of the system clock is to synchronize major operations such as the setup of addresses, and the reading and writing of data. In this case, the \$C040 line which corresponds to the UTILSTB is active low only when the proper address is placed and the decoder and the phase 0 system clock is high. Since this is a 1 MHz clock, it is high only for $\frac{1}{2}$ μ sec. Therefore, UTILSTB can be active low only for a $\frac{1}{2}$ μ sec duration, while it is being addressed. Now you know why accessing the \$C040 location (either from BASIC or machine language) results in a brief $\frac{1}{2}$ μ sec down-going pulse.

Observe that the UTILSTB line is distributed directly to the game port socket, with no intervening hardware. The on-board I/O decoder chip serves dual roles as an interface: it decodes for the UTILSTB signal and also buffers the signal to normal LSTTL fan-out. Naturally, we can use the UTILSTB signal to create stable high or low signals by employing an external flip-flop, as done in earlier experiments.

Two other points: the UTILSTB line should be activated by a read command not a write command—e.g., a PEEK, not a POKE from BASIC. If you access UTILSTB with a POKE, then two negative pulses will be output on the line instead of one.

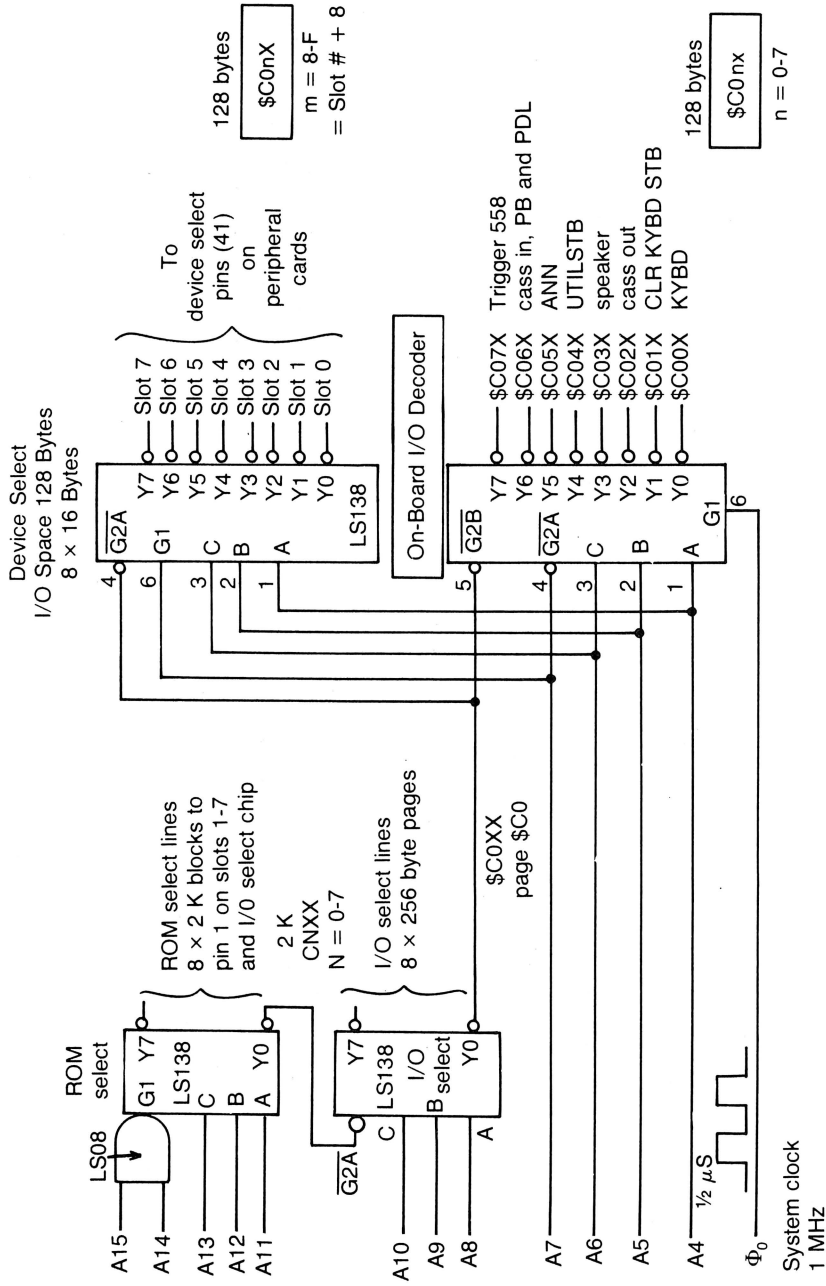
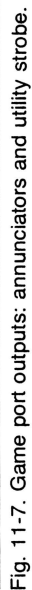


Fig. 11-6. Recapitulation of the address decoding circuitry, with additional details on the decoding of page \$C0.



This is because of certain technicalities in system timing. Also, you should understand why any location in the 16 byte range from \$C040-\$C04F will activate the UTILSTB; the answer is because this line is not decoded any further after it exits the Y4 pin of the on-board decoder.

Unlike the UTILSTB line, the annunciator line \$C05x is not distributed directly to the game connector. Instead, it is further decoded using the LS259 Addressable latch. This latch is activated by the \$C05x (Y5) output of the on-board decoder. The upper four outputs of the latch—Q4 to Q7—are used for the AN0 to AN3 lines. Any one of these lines can be selected by the right input one the three address inputs A1 to A3. These will have a binary number in the range 100 to 111, for a total of four possibilities. For any one of the annunciators selected, address line A0 will determine whether it is to be high or low (on/off). The short truth table at the bottom of Fig. 11-7 illustrates this action of A0 as the on/off bit. Review the functions of the addressable latch mode of LS259 operation, if necessary, to verify the above.

Note that since the LS259 is a latch, it will hold the data in a stable fashion, so that a high or low on a given output will remain in that state unless changed. Also, observe that as far as the programmer is concerned, it looks as if there are four pairs of on/off locations, each corresponding one of the addresses in the range \$C058-\$C05F, with the even numbered locations turning the corresponding annunciator off (A0=0), and the odd numbered locations turning the corresponding annunciator on (A0=1).

What about the lower four outputs of the LS259? These are also treated as four pairs of on/off location, the only difference being that the A3 line is low. Again A0 acts as the on/off switch for each of these four lines. As to the function of these lines, they are used as screen soft switches. These are locations that can be accessed from BASIC to set the screen display, and may be familiar to you already: they serve to set LORES/HIRES/text modes, mixed text and graphics, all text or graphics, etc.

Inputs

Again we'll concentrate on the signals used on the game connector. Refer to Fig. 11-8. You'll note that there are three types of inputs to the computer data bus: one cassette input (CIN), three pushbutton inputs (PB0-PB2), and four analog or game paddle inputs (PDL0-PDL3).

Before getting into the circuitry involved, one important fact that should be mentioned. The status of all of these single bit inputs is placed on a single data bus line: the D7 or most significant bit of the system data bus. This means that when the corresponding input is high, this bit is set or equal to one. Therefore, regardless of the state of the other seven data lines (D0-D6), the number on the data bus will be 1000 0000 or greater when D7 is high. This corresponds to decimal 128 or more. This is why such locations are PEEKed as to whether they are > 127 or < 128 to determine their high or low status.

Now, to explain the applications input circuitry, we'll work backwards, starting from the D7 data line.

The LS251 Data Selector. This chip decodes for addresses in the range \$C060-\$C06F, and is activated at its strobe' pin by the active low \$C06x line. There are eight inputs to the data selector/multiplexer—from cassette input, push-buttons and paddles—and three address lines to select one of these inputs. The non-inverted three-state output is connected bit D7 of the system data bus.

The sharp-eyed among you may notice that the active low \$C06x signal allows for sixteen locations, even though there are only eight inputs to this demultiplexer. What is going on? The explanation is clear if you note that there is no A3 address line connected to this device. This means that as far as this LS251 is concerned, addresses 0000-0111 look the same as addresses 1000-1111. This means that the range of on-board input can be read twice. That is, the range \$C060-\$C067 is repeated, in the same sequence, in the range \$C068-\$C06F. (This was mentioned briefly in Table 11-3).

Cassette Input. While not connected to the

game port, this input to the computer is mentioned because it is part of the BDIS system. The cassette input op-amp circuitry is not detailed. Suffice it to say that this input is connected to the D0 pin on the LS251, and corresponds to locations \$C060 or \$C068. We use it as the fourth pushbutton input.

Pushbutton Inputs. These three lines are connected directly to the LS251 from the game connector. The primary addresses are \$C061-\$C063. They can be driven by any two-state device—a mechanical switch, TTL output, etc. The LS251 serves as a decoder and also buffers these signals, that is, normalizes them to LSTTL fan-out current levels.

Analog or Paddle Inputs. Four output lines from the 558 quad one shot (or timer as it is often called), complete the set of game port inputs we'll discuss. The primary locations of these inputs are \$C064-\$C067. Usually, a variable resistor is connected to one of these inputs. A single such input can be used as a game paddle input, a pair can be used as a joystick input.

Figure 11-8 shows that the 558 quad one-shot device is triggered by the \$C070 line off the on-board I/O decoder. Again, any address in the \$C070-\$C07F range will trigger this one shot. Note that all timers on the chip are triggered simultaneously. On each timer each timer output will appear a square wave, the duration of which depends on the time constant (TC) of the RC network attached to the timing input (A thru D). The pulse width of the output square wave is equal to RC. The width of the square wave is measured by a software timing loop in the Apple Monitor, called PREAD for paddle read.

These analog inputs provide the basis for single bit A/D conversion with fairly modest means and we'll take a look at this subject next.

You should look over Table 11-4. It provides a capsule summary of the on-board I/O decoding circuitry covered in this section.

SIMPLE APPLICATIONS CIRCUITS

Here we'll describe the standard setup of the Apple game paddles, including the circuitry and Monitor paddle reading software.

Standard Analog Input Hardware and Software

The analog or paddle inputs on Apple's game port allow for single bit A/D conversion. The hardware basis for this, with which you are already familiar, is repeated in Fig. 11-9A. The on-board resistor and capacitor are attached to the timing input of one of the 558 one shots. As shown, a potentiometer is connected to the respective game port paddle input as a variable resistor. That is, one end is connected to the +5 volt supply, the wiper is connected to the paddle input, and the other end is left free. (This potentiometer is connected as a variable resistor, and so technically is a rheostat).

That portion of the pot resistance that appears in the circuit depends upon the position of the wiper, which in turn depends, in the usual case, on the position of the game paddle dial. This in-line resistance we'll call R_p , as marked in the figure.

As you know, in the one shot the square wave output's duration or pulse width (PW) depends on the external RC time constant. In this case the TC equals $(R_1 + R_p)C_1$, where R_1 and C_1 are on the Apple main board. If the pot is set to zero, then the time constant will be $R_1 \times C_1 = .022 \mu F \times 100 \text{ ohms}$ or $2.2 \mu \text{sec}$. If the 150 K pot (as comes with the Apple game paddle) is set at maximum, then the TC will be about $R_p \times C_1 = 150 \text{ K} \times .022 \mu F$ or $3300 \mu \text{sec}$ (3.3 millisecc).

So you have a square wave, which appears as a signal on data bus line D7, and which remains at logic high for some duration, depending on the value of the game paddle pot. How is this duration measured and converted into some meaningful value?

The answer is in the PREAD routine that is part of Apple's Monitor. If you turn to the Monitor listing in your Apple Reference Manual beginning at \$FB1E you'll find the PREAD routine. For those of you who know little or nothing about assembly language, the explanation of the routine is as follows.

The calling program places the number of paddle to be read in the Apple's 6502 X-register. Microprocessors have temporary storage locations, called registers and accumulators, that can be di-

Table 11-4. Game Port Memory: Pinout, Chips and Memory Locations.

GPIN#	Name	Address		Read or Write	Chips
		Decimal	Hex		
1, 8	+5, GND	-	-	-	-
-	Cass. In	49,248	\$C060	R	op-amp CIRCUIT
2	PB0	49,249	\$C061	R	LS251 MUX, 8T28*
3	PB1	49,250	\$C062	R	LS251 MUX, 8T28*
4	PB2	49,251	\$C063	R	LS251 MUX, 8T28*
5	UTILSTB	49,216	\$C040	R	direct off on-bd I/O decoder
6	PDL0	49,252	\$C064	R	558,LS251 MUX, 8T28
7	PDL2	49,254	\$C066	R	558,LS251 MUX, 8T28
10	PDL1	49,253	\$C065	R	558,LS251 MUX, 8T28
11	PDL3	49,255	\$C067	R	558,LS251 MUX, 8T28
12	AN3	49,246/7	\$C05E,F	R or W	LS259 Latched deMUX
13	AN2	49,244/5	\$C05C,D	R or W	LS259 Latched deMUX
14	AN1	49,242/3	\$C05A,B	R or W	LS259 Latched deMUX
15	AN0	49,240/1	\$C058,9	R or W	LS259 Latched deMUX
9,16		(NC no connection)			

* Note: The 8T28 is a bus transceiver similar to the LS245, and is used to buffer the system data bus. Since it is a quad rather than octal device, two are used.

rectly accessed by assembly language. In Apple's 6502, two registers are present, called X and Y; each are 8 bits wide. Let's assume that the calling program is BASIC (some target game for example), and the command "X=PDL(0)" is encountered at some point in this program. The BASIC interpreter, when encountering this instruction, would first load the X-register with the value 0, and then jump to the PREAD routine in the monitor.

The first thing that happens in PREAD is that the 558 timer is triggered by accessing \$C070. Then the Y-register, which is to serve as a counter, is set to zero. Then the particular paddle location is evaluated: the logic level on the D7 line, taken off the noninverted output of the LS251 multiplexer, is read into the 6502's accumulator. The routine then checks to see if this value is zero or nonzero. If nonzero, this means that the square wave pulse is still high, and the counter (Y-register) is incremented.

Next the routine checks to see if the counter has reached zero. If not, the count continues, until the 558 line being measured "times out", leaving the final value of the paddle in the Y-register. This

value will fall in the range of 0-255, and can be used by the program for a variety of purposes.

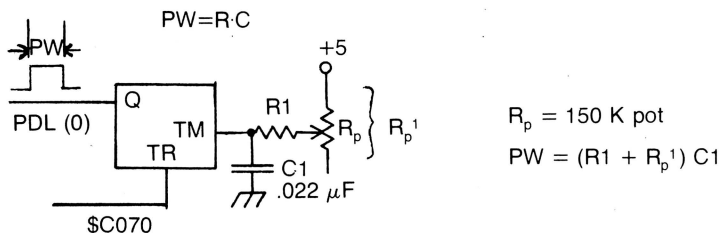
What happens if the time constant of the RC is greater than the maximum loop time in PREAD. Then in such a case, PREAD will keep counting up to 255, and then increment to zero: when an 8 bit counter reaches \$FF (255), incrementing it by one gives you \$00. Therefore, if you've counted up to zero, it means that you've reached the maximum range of the 8-bit counter, namely \$FF or 255 decimal. At this point PREAD decrements the counter by one (back to \$FF) and returns to the calling program (RTS return from subroutine) with the value 255 in the Y-register.

Since each loop through PREAD takes about 12 μ sec, the max loop time for looking at the paddle is

$$\text{PREAD, max loop time} = 255 \times 12\mu\text{sec/loop} = 3060 \mu\text{sec or } 3.06 \text{ millisec}$$

Note that the minimum time between triggering the 558 and exiting the loop is about 13 μ sec. This means that any time constant less than 13 msec will

A



B

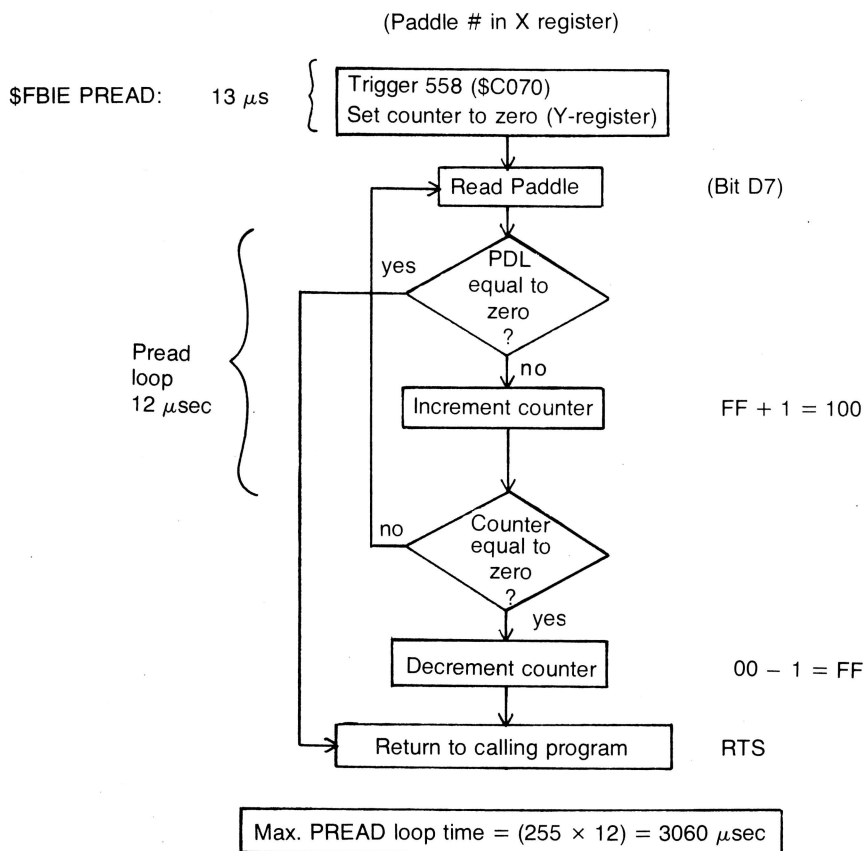


Fig. 11-9. 558 timer with RC circuit and flowchart of PREAD: standard gear for paddle reading.

return to zero value, and any $TC > 3.06$ millisecc will return 255.

In other words values of the RC time constant between about $13\ \mu\text{sec}$ and $3060\ \mu\text{sec}$ will return values of 1 to 254. TCs outside this range result in less useful values from PREAD: 0 and 255 only tell you that the TC falls into a low or high range, and in this sense they are not really translatable into exact time units.

Still another way of expressing the above is to say that the dynamic range of PREAD is only 254 to 1, or somewhat better than two orders of magnitude.

The maximum TC of the Apple paddle circuitry (3.3 msec), when the pot is turned up to 150 K, exceeds the maximum PREAD loop time (3.06 msec). That is, there is a certain *dead space* at the end of the pot travel, with the normal 150 K game paddle attached. This is acceptable, as some downside variation in pot resistance due to normal tolerance in manufacture is to be expected. If the pot turns out to have a less than 150 K max. resistance, then there is a small safety margin, and you'll still get a 255 value when it is turned to maximum.

Finally, another aspect of the variable resistor and PREAD combination is that the greater the resistance, the longer it takes PREAD to read it. This means that a series of readings on a quickly changing resistor will be separated by unequal time intervals. This is important to know if you want to store the readings and later display them as a function of time.

What is really being measured by PREAD? How far away you are from zapping a space monster, or eating a ghost or power pellet?

Obviously, you are measuring an RC time constant and anything that changes that time constant changes the value returned by PREAD. In fact, you don't have to use a variable resistor at all. Instead, you may want to measure physical quantities (light, temperature, pressure, strain, physico-chemical properties like pH, etc.) by employing not a resistor but some other component (photocell, thermistor, etc.) in which resistance changes in response to that physical quantity. All such components (including the pot, which in a sense measures

angular position) fall under the heading of *resistive transducers*.

What about the problem of interfacing variable resistances or with nonstandard maximum values (those other than 150 K max value) to the game port? While this can usually be dealt with easily, you might wonder if such an exercise is really useful. The fact is, if you can interface some arbitrary pot value so that it returns a range of 0 to 255 using PREAD, you can most likely interface any resistance. This includes resistive transducers which you would want to use in laboratory measurement and similar A/D applications. You'll see how this is done next.

Matching Time Constants

To hook up any resistive transducer to one of Apple's analog inputs you must match the maximum hardware time constant with the maximum PREAD value. This means that the TCmax of the RC network attached to the timing input should usually be about equal to the PREADmax time of $3.06\ \mu\text{sec}$.

But there are times when the RC time constant max value—TCmax—winds up to be something more or less than the PREAD max of 3.06 msec. At other times a value slightly less may be acceptable. The question is, just how do you adjust the TC for an arbitrary resistance? By adding external capacitance!

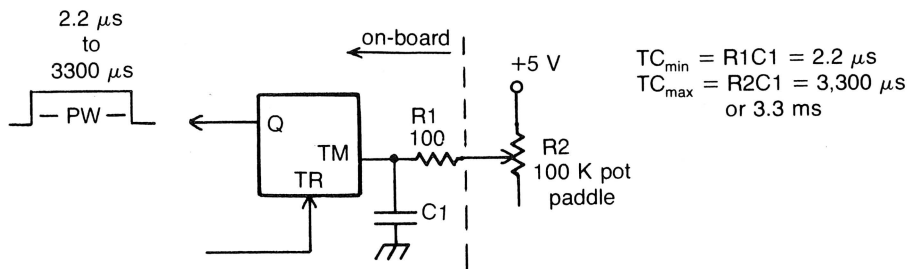
We are matching the hardware to the monitor PREAD routine. Normally, in a from-scratch I/O project, it would be the other way around. But PREAD is part of the built-in firmware, is readily accessed from built-in BASIC commands, and does its job efficiently. So its loop times are the basis for our simple hardware (RC time constant) design decisions.

In Fig. 11-10A the usual setup of the Apple paddle 150 K pot connected to the on-board timer is shown. Again the important times are:

Hardware time	
constant (RC)	= 2.2 to $3,300\ \mu\text{sec}$ range
Monitor PREAD	
loop time	= 13 to $3,060\ \mu\text{sec}$ range

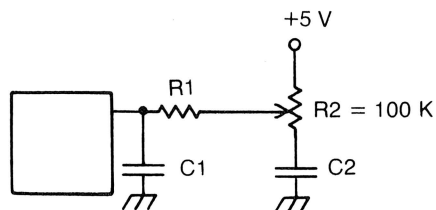
Assume that the pot resistor is less than 150

A

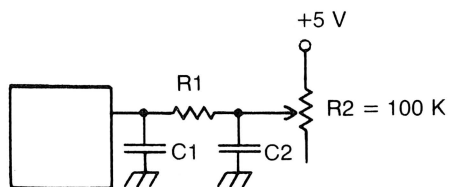


PREAD: 13 μ s to 3,060 μ s

B



C



$$C2 \geq \frac{\text{PREAD MAX}}{R_{\text{pot}}} - C1$$

Fig. 11-10. Additional capacitance is necessary for potentiometers less than 150 K. Best placed as shown in (C).

K. Then by adding capacitance, we can make sure that TCmax still approaches PREADmax by adding a capacitor. The first problem is where to add it in the circuit of Fig. 11-10A. Two alternatives are shown in Fig. 11-10B and C. In Fig. 11-10B the capacitor is added in line with the pot. This is not really correct, though it is often seen in "add your own joystick" type articles.

A better way is to attach the extra capacitance, which we'll call C2, as shown in Fig. 11-10C. Remember that to add capacitors you place them in parallel. The configuration in Fig. 11-10C places C2 almost directly in parallel with the on-board capacitor, C1. There is no intervening variable resistance as in Fig. 11-10B, so this arrangement is preferred. (The small 100 ohm current limiting on-board resistor has little effect over most of the range of RC, so we can ignore it).

The calculation for the added capacitor, C2 is

$$C2 = (PREAD_{max} / R_{pot}) - C1$$

or

$$C2 = (3.06 \text{ ms} / R_{pot}) - .022 \mu\text{F}$$

where R_{pot} is in Kohms
C2 is in μF

Figure 11-11 shows a calculation for a 100 K pot. Figure 11-11A indicates the normal situation with the 150 K pot, with a .24 msec difference between PREADmax and TCmax; this is equivalent to about 10.9 Kohms at the upper end of pot travel, i.e., dead space. There is an 8% margin of tolerance in this difference.

Another formula we could use is:

$$C2 = ((150 \text{ K} / R_{pot}) \times .022) - .022$$

In Fig. 11-11B we simply plug in the values for the equation for added capacitance. C2 works out to about 0.0086 μF if we want TCmax to equal PREADmax. To give a little margin of dead space—that is, to assure we will always reach a 255 value from PREAD at the end of pot travel—you can boost up C2 a bit. A value of 0.01 μF could be used, because it is a common value. The resulting 16% safety margin may leave too much dead space, however.

Figure 11-11C provides the solution. Just add the .1 and .01 capacitors (caps A and B shown to the right) in series. Remembering the rule for series additions, you'll come up with a net value of .91 μF . This is a more reasonable safety margin for C2, enough (about 6%) but not too much.

Figure 11-11C also shows how to increase the value of C2 when needed: just add small values of capacitance in parallel.

Matching Non Standard Resistances

Try the TC matching technique out for yourself by getting an assortment of small disc capacitors and a pot in the 50 to 100 Kohm range. Make sure to measure the pot yourself with an ohmmeter. Don't take the face value on the package, as there is often considerable variability between specified and actual maximum resistance, as much as 20%. Then work out the solution for C2 using the formula.

Using the tiny program in Listing 11-1, find out how good your calculated value for C2 really is. Increase or decrease the value of C2 as suggested in Fig. 11-11C, so that you have just the right amount of dead space. How far away were you from the calculated value?

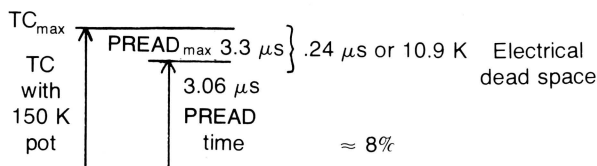
Question: By changing things around a bit, can you use this scheme as a crude measure of capacitance?

Question: For pot values greater than 150 K, how do you match the time constant? Hint—parallel resistances.

Now that you know how to match time constants to PREAD, let's look at a rational way of interfacing a joystick to your computer.

Joystick Interfacing Techniques

Apple's built-in analog timer circuit and PREAD routine makes interfacing resistive transducers much easier than on some other machines. You need only match time constants and you're home free. Right? Not quite. Matching a joystick to a graphics screen display require something more than matching time constants alone. Correctly choosing the RC values is the important first step,

A

$$TC \approx R_{pot} (C1 + C2)$$

$$\left\{ \begin{array}{l} TC_{max} = 3.06 \\ C1 = .022 \ \mu F \end{array} \right. \quad \left(\begin{array}{l} PREAD, \text{ max} \\ \text{min acceptable} \end{array} \right)$$

B

$$C2 \geq \frac{TC_{max}}{R_{pot}} - C1$$

$$C2 \geq \frac{3.06 \text{ ms}}{100 \text{ k}} - .022 \quad \text{or}$$

$$C2 \geq .0086 \ \mu F$$

or $.01 \ \mu F$

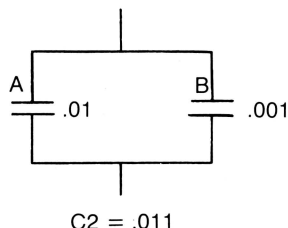
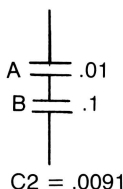
C

Fig. 11-11. (A) Dead space on paddle is due to a time constant somewhat longer than PREADmax. (B) Formula for calculating additional capacitance in cases where $R_p < 150 \text{ K}$. (C) Fine-tuning the value of C2.

```

4  REM -----
5  REM  PDL TEST
6  REM -----
7  REM
10 HOME
20 P = 0
30 PRINT PDL (P)
40 GOTO 30

```

Listing 11-1. Simple program for reading a paddle value.

true, but you need some software to interpret the values obtained. This will become clear as we run through a joystick interfacing example.

Figure 11-12 shows the joystick we'll be working with—one with two 100 K pots. A Radio Shack 271-1705 or the equivalent will do. Assume we have a graphics application; one pot will be the X-pot and the other the Y-pot, connected with the orientation shown. The steps in interfacing are:

1. Choosing the values of C_x and C_y .
2. Scaling the PREAD values to the screen display.

For simplicity's sake, let's assume a LORES

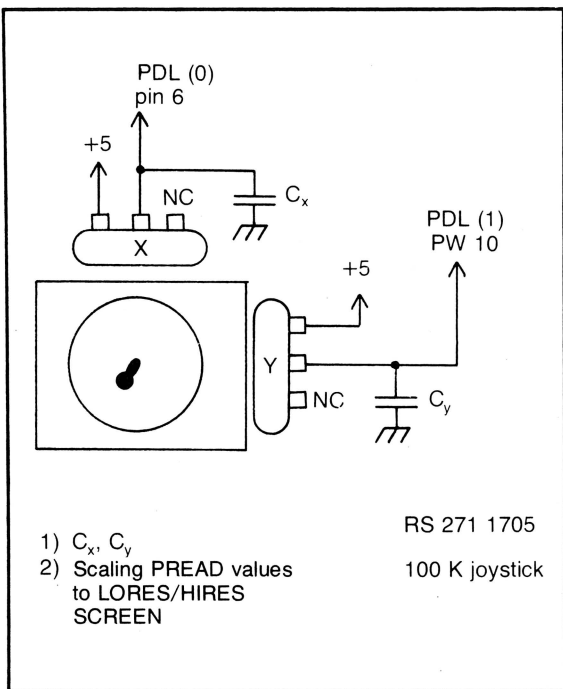


Fig. 11-12. Statement of the basic factors involved in interfacing a joystick. 100 K used as an example.

(low resolution graphics screen) application, and that you've already calculated value for C_x and C_y , so that you have a full excursion of 0 to 255 for the X-pot and Y-pot. Now what software is necessary to give you proper SCREEN excursion?

Figure 11-13 illustrates what is involved in matching the PREAD value returned to the screen position. The circle in 11-13A represents the excursion of the pot. Point A and B represent the X-pot travel, with PX (the value returned by PREAD from the X-pot) going from 0 to 255. Likewise, points C and D represent the extremes of PY, the value for the Y-pot returned by PREAD.

Note that there are other points both on and within this circle. Point E in the right lower quadrant of travel, might have a value around 200,200 for PX,PY. Point F in the center of pot travel would have PX,PY values of 127,127, assuming perfect linearity. Also, there are "corners" of travel, namely at the *extreme* points of joystick excursion in each *quadrant*: left upper and lower points (LUP

and LLP), and the right upper and lower points (RUP and RLP).

Figure 11-13B shows the square shape of the screen, be it TEXT, LORES or HIRES. The points of interest are the corners which are labeled as LUC, LLC, RUC, and RLC. They might have the PX,PY values shown in the figure.

How do you match joystick excursion to screen position? That is, how should these two shapes—circle and square—be reconciled? The answer lies in how the screen corners are related to the quadrant points on the joystick excursion circle.

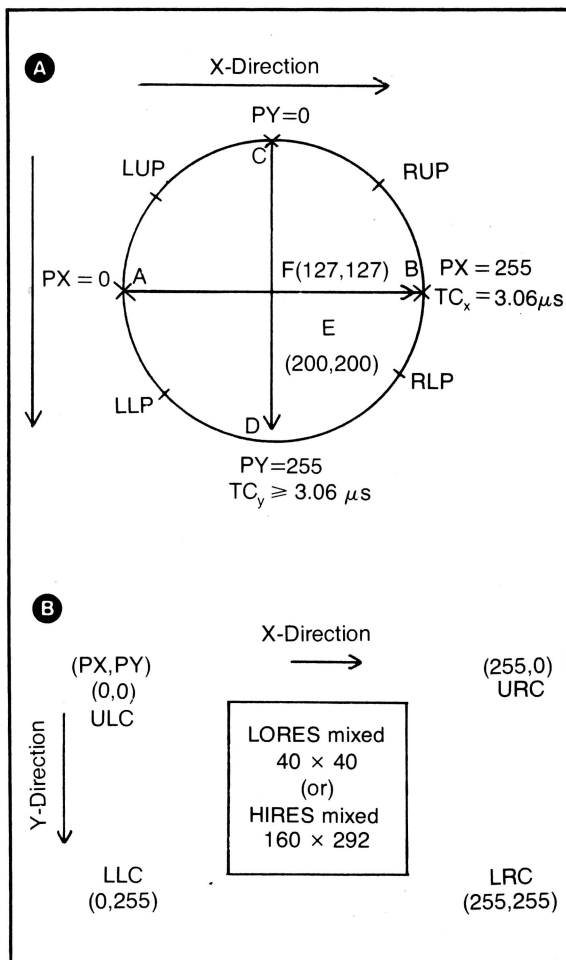


Fig. 11-13. (A) Points on the joystick excursion circle are represented by pairs of paddle values, PX and PY. (B) Screen corners should correspond to the points on the circle. (LUP, etc.).

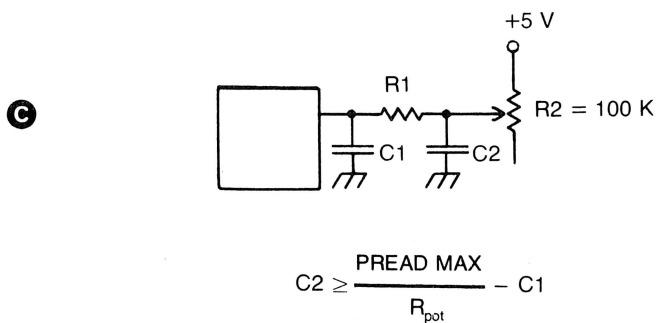
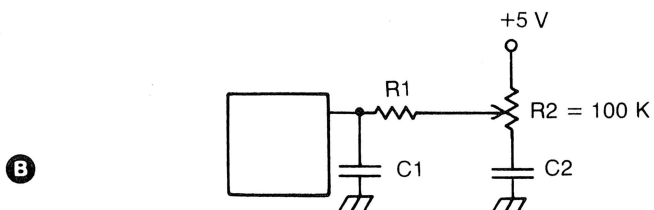
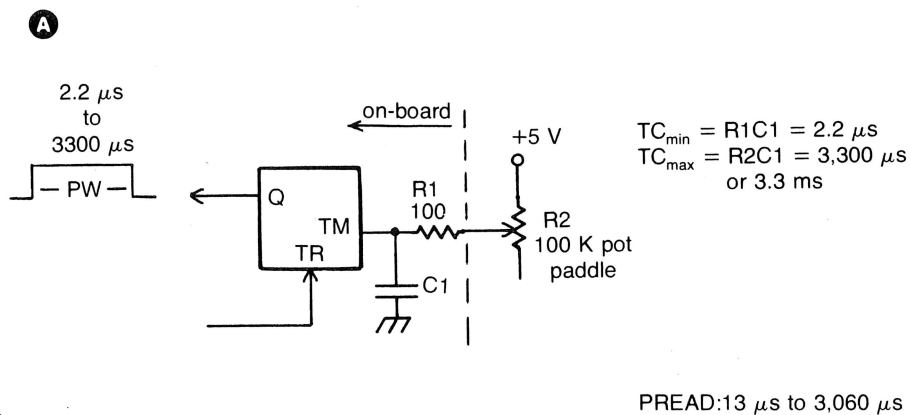


Fig. 11-10. Additional capacitance is necessary for potentiometers less than 150 K. Best placed as shown in (C).

K. Then by adding capacitance, we can make sure that TCmax still approaches PREADmax by adding a capacitor. The first problem is where to add it in the circuit of Fig. 11-10A. Two alternatives are shown in Fig. 11-10B and C. In Fig. 11-10B the capacitor is added in line with the pot. This is not really correct, though it is often seen in “add your own joystick” type articles.

A better way is to attach the extra capacitance, which we’ll call C2, as shown in Fig. 11-10C. Remember that to add capacitors you place them in parallel. The configuration in Fig. 11-10C places C2 almost directly in parallel with the on-board capacitor, C1. There is no intervening variable resistance as in Fig. 11-10B, so this arrangement is preferred. (The small 100 ohm current limiting on-board resistor has little effect over most of the range of RC, so we can ignore it).

The calculation for the added capacitor, C2 is

$$C2 = (PREAD_{max} / R_{pot}) - C1$$

or

$$C2 = (3.06 \text{ ms} / R_{pot}) - .022 \mu\text{F}$$

where R_{pot} is in Kohms
C2 is in μF

Figure 11-11 shows a calculation for a 100 K pot. Figure 11-11A indicates the normal situation with the 150 K pot, with a .24 msec difference between PREADmax and TCmax; this is equivalent to about 10.9 Kohms at the upper end of pot travel, i.e., dead space. There is an 8% margin of tolerance in this difference.

Another formula we could use is:

$$C2 = ((150 \text{ K} / R_{pot}) \times .022) - .022$$

In Fig. 11-11B we simply plug in the values for the equation for added capacitance. C2 works out to about 0.0086 μF if we want TCmax to equal PREADmax. To give a little margin of dead space—that is, to assure we will always reach a 255 value from PREAD at the end of pot travel—you can boost up C2 a bit. A value of 0.01 μF could be used, because it is a common value. The resulting 16% safety margin may leave too much dead space, however.

Figure 11-11C provides the solution. Just add the .1 and .01 capacitors (caps A and B shown to the right) in series. Remembering the rule for series additions, you’ll come up with a net value of .91 μF . This is a more reasonable safety margin for C2, enough (about 6%) but not too much.

Figure 11-11C also shows how to increase the value of C2 when needed: just add small values of capacitance in parallel.

Matching Non Standard Resistances

Try the TC matching technique out for yourself by getting an assortment of small disc capacitors and a pot in the 50 to 100 Kohm range. Make sure to measure the pot yourself with an ohmmeter. Don’t take the face value on the package, as there is often considerable variability between specified and actual maximum resistance, as much as 20%. Then work out the solution for C2 using the formula.

Using the tiny program in Listing 11-1, find out how good your calculated value for C2 really is. Increase or decrease the value of C2 as suggested in Fig. 11-11C, so that you have just the right amount of dead space. How far away were you from the calculated value?

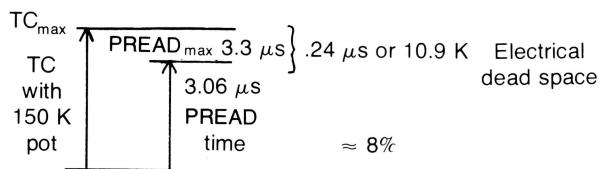
Question: By changing things around a bit, can you use this scheme as a crude measure of capacitance?

Question: For pot values greater than 150 K, how do you match the time constant? Hint—parallel resistances.

Now that you know how to match time constants to PREAD, let’s look at a rational way of interfacing a joystick to your computer.

Joystick Interfacing Techniques

Apple’s built-in analog timer circuit and PREAD routine makes interfacing resistive transducers much easier than on some other machines. You need only match time constants and you’re home free. Right? Not quite. Matching a joystick to a graphics screen display require something more than matching time constants alone. Correctly choosing the RC values is the important first step,

A

$$TC \approx R_{pot} (C_1 + C_2)$$

MS K μF

$$\begin{cases} TC_{max} = 3.06 \\ C_1 = .022 \mu F \end{cases} \quad \left(\begin{array}{l} \text{PREAD, max} \\ \text{min acceptable} \end{array} \right)$$

B

$$C_2 \geq \frac{TC_{max}}{R_{pot}} - C_1$$

$$C_2 \geq \frac{3.06 \text{ ms}}{100 \text{ k}} - .022 \quad \text{or}$$

$$C_2 \geq .0086 \mu F$$

or .01 μF

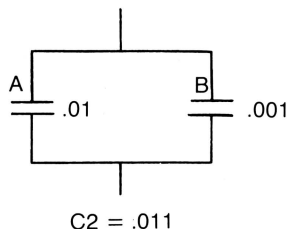
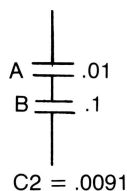
C

Fig. 11-11. (A) Dead space on paddle is due to a time constant somewhat longer than PREAD_{max}. (B) Formula for calculating additional capacitance in cases where $R_p \leq 150 \text{ K}$. (C) Fine-tuning the value of C_2 .

```

4  REM -----
5  REM  PDL TEST
6  REM -----
7  REM
10 HOME
20 P = 0
30 PRINT PDL (P)
40 GOTO 30

```

Listing 11-1. Simple program for reading a paddle value.

true, but you need some software to interpret the values obtained. This will become clear as we run through a joystick interfacing example.

Figure 11-12 shows the joystick we'll be working with—one with two 100 K pots. A Radio Shack 271-1705 or the equivalent will do. Assume we have a graphics application; one pot will be the X-pot and the other the Y-pot, connected with the orientation shown. The steps in interfacing are:

1. Choosing the values of C_x and C_y .
2. Scaling the PREAD values to the screen display.

For simplicity's sake, let's assume a LORES

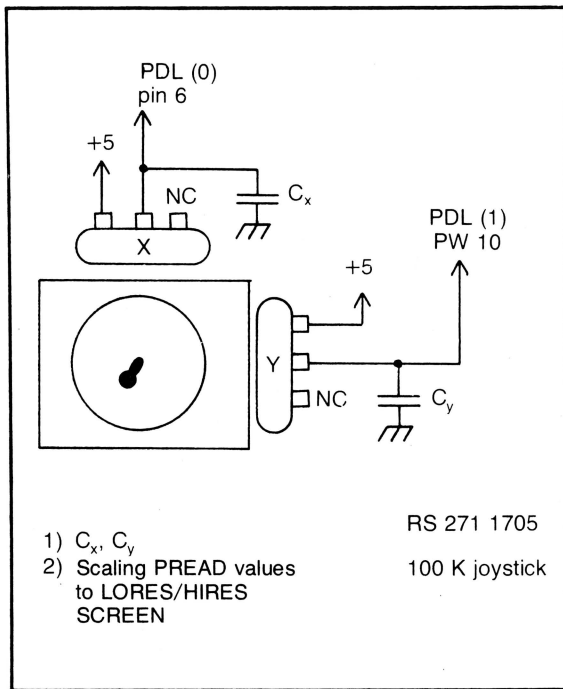


Fig. 11-12. Statement of the basic factors involved in interfacing a joystick. 100 K used as an example.

(low resolution graphics screen) application, and that you've already calculated value for C_x and C_y , so that you have a full excursion of 0 to 255 for the X-pot and Y-pot. Now what software is necessary to give you proper SCREEN excursion?

Figure 11-13 illustrates what is involved in matching the PREAD value returned to the screen position screen position. The circle in 11-13A represents the excursion of the pot. Point A and B represent the X-pot travel, with PX (the value returned by PREAD from the X-pot) going from 0 to 255. Likewise, points C and D represent the extremes of PY, the value for the Y-pot returned by PREAD.

Note that there are other points both on and within this circle. Point E in the right lower quadrant of travel, might have a value around 200,200 for PX,PY. Point F in the center of pot travel would have PX,PY values of 127,127, assuming perfect linearity. Also, there are "corners" of travel, namely at the *extreme* points of joystick excursion in each *quadrant*: left upper and lower points (LUP

and LLP), and the right upper and lower points (RUP and RLP).

Figure 11-13B shows the square shape of the screen, be it TEXT, LORES or HIRES. The points of interest are the corners which are labeled as LUC, LLC, RUC, and RLC. They might have the PX,PY values shown in the figure.

How do you match joystick excursion to screen position? That is, how should these two shapes—circle and square—be reconciled? The answer lies in how the screen corners are related to the quadrant points on the joystick excursion circle.

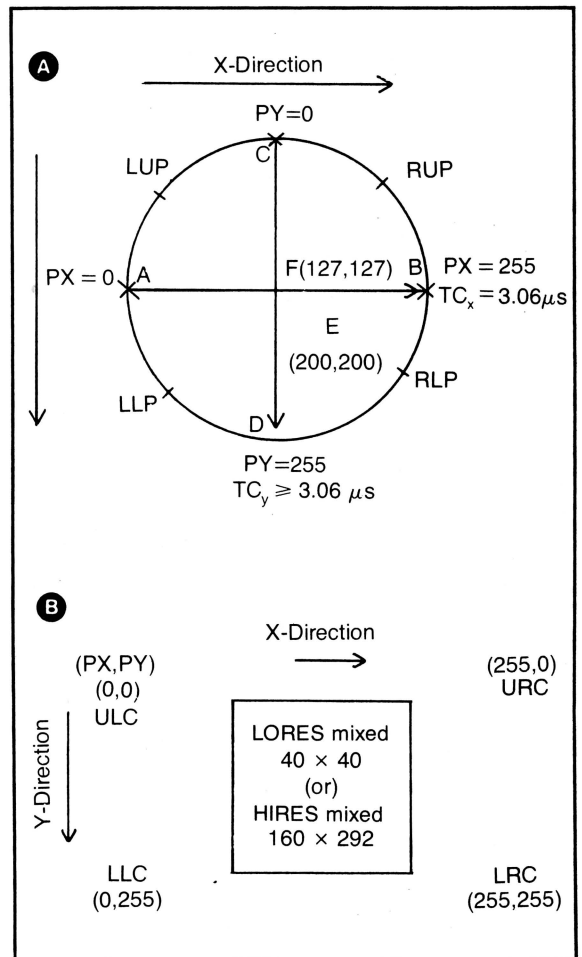


Fig. 11-13. (A) Points on the joystick excursion circle are represented by pairs of paddle values, PX and PY. (B) Screen corners should correspond to the points on the circle. (LUP, etc.).

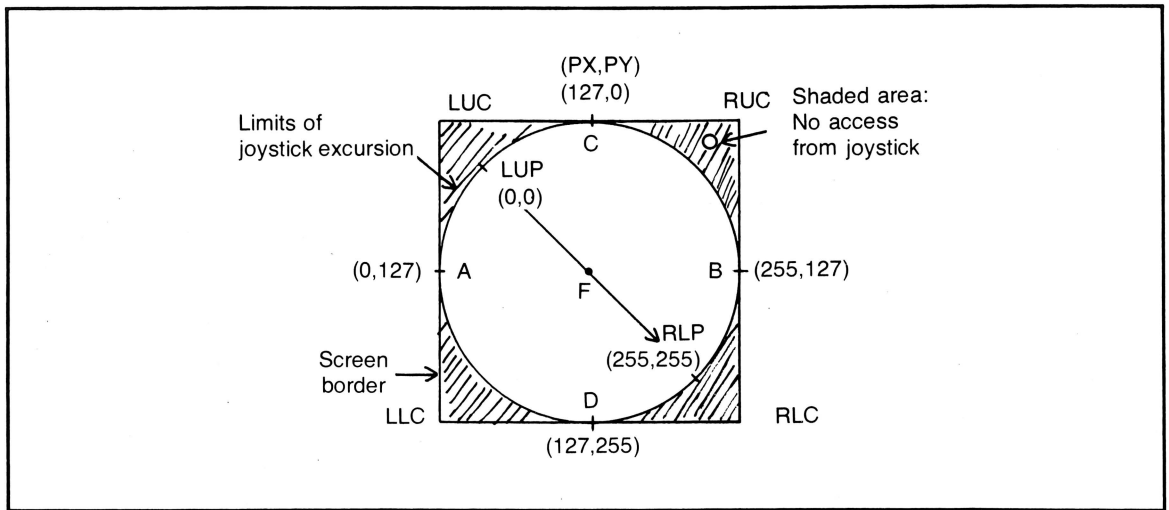


Fig. 11-14. In this arrangement, the joystick can never reach the corners of the screen.

One possible arrangement is illustrated in Fig. 11-14. If you were to make points A to D on the joystick circle line up on the respective sides of the screen square, then you would have the circle inside the square, as shown. (This would be done by appropriate scaling in software, as will be explained shortly). However, there is an obvious problem with the arrangement shown in Fig. 11-14; the corners of the screen will be inaccessible to the joystick! This is indicated by the shaded area, and is obviously not a desirable situation.

Instead, we'll set the scaling factors in the software to make the joystick excursion fall outside the screen range of coordinate values. Then the relationship between joystick points and screen corners would be something like that illustrated in Fig. 11-15. By introducing the right scale factors, the PX and PY values returned can be converted to, for instance, LORES screen coordinates. These would fall in the range of 0 to 39 for a mixed LORES text display.

More precisely, we start with the assumption

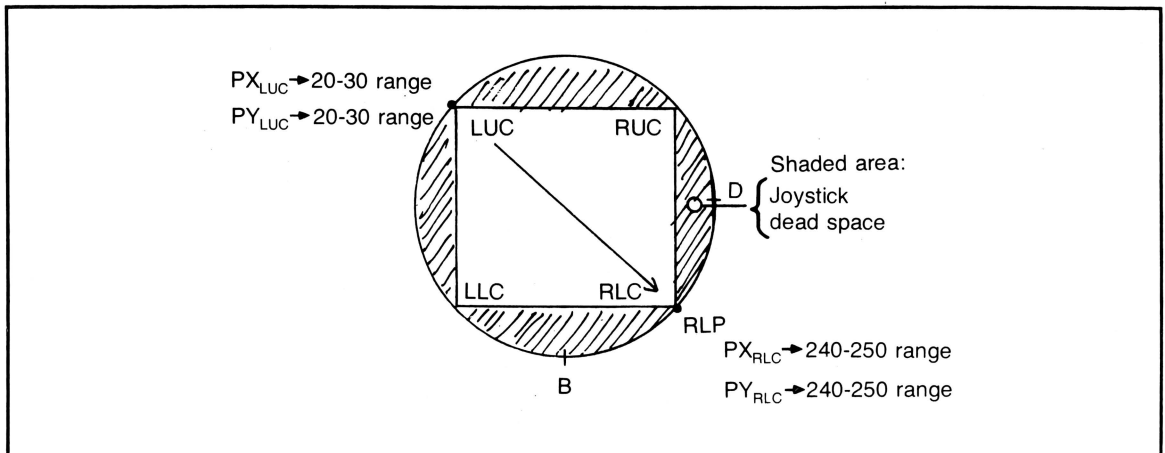


Fig. 11-15. When properly scaled by software scaling factors, the PX and PY values can be made to correspond to the screen corners. Joystick dead space is dealt with in software.

that the time constants for the X and Y circuits are chosen to give maximum values for X-pot and Y-pot excursion (points B and D). When the joystick is positioned in the LUC of the screen, then the value returned for PX and PY will be something more than the minimum. That is PX,PY will be more than 0,0—probably in the range of 20 to 30. Likewise, the value of PX and PY at the RLC of the screen will be somewhat below maximum, probably in the range of 240 to 250. Remember that these are the important points, the ones whose PX,PY values we must use to write the proper scaling factors in the software equations.

Naturally, with the arrangement shown in Fig. 11-15, we'll have some dead space below and above the PX and PY minimum and maximum values of 20-30 and 240-250. This dead space, indicated by the shaded areas, has to be dealt with in software, as you'll see in the example to follow.

LORES/HIRES JOYSTICK PROJECT

You have a cheap 100 K joystick. How do you interface it to the Apple game port?

Start by measuring the two pots to find their real value. Figure 11-16 indicates the procedure for one of the pots, which we'll call the X-pot. Figure 11-17 shows the same sequence for the Y-pot. By convention, we'll attach the X-pot to PDL(0)-pin 6, and the Y-pot to PDL(1)-pin 10 on the game connector. The values given in this project are real values obtained when this demonstration was originally tested out. Let's now determine the RC circuit and appropriate scaling factors for these two pots.

1. As shown in Fig. 11-16A, the value of R_x and C_x must be determined. Measure the max value of the X-pot, which we'll call R_x . (This 100 K pot actually turned out to be 85 K! Definitely, this is a real-life example). Then determine the value of C_x , using

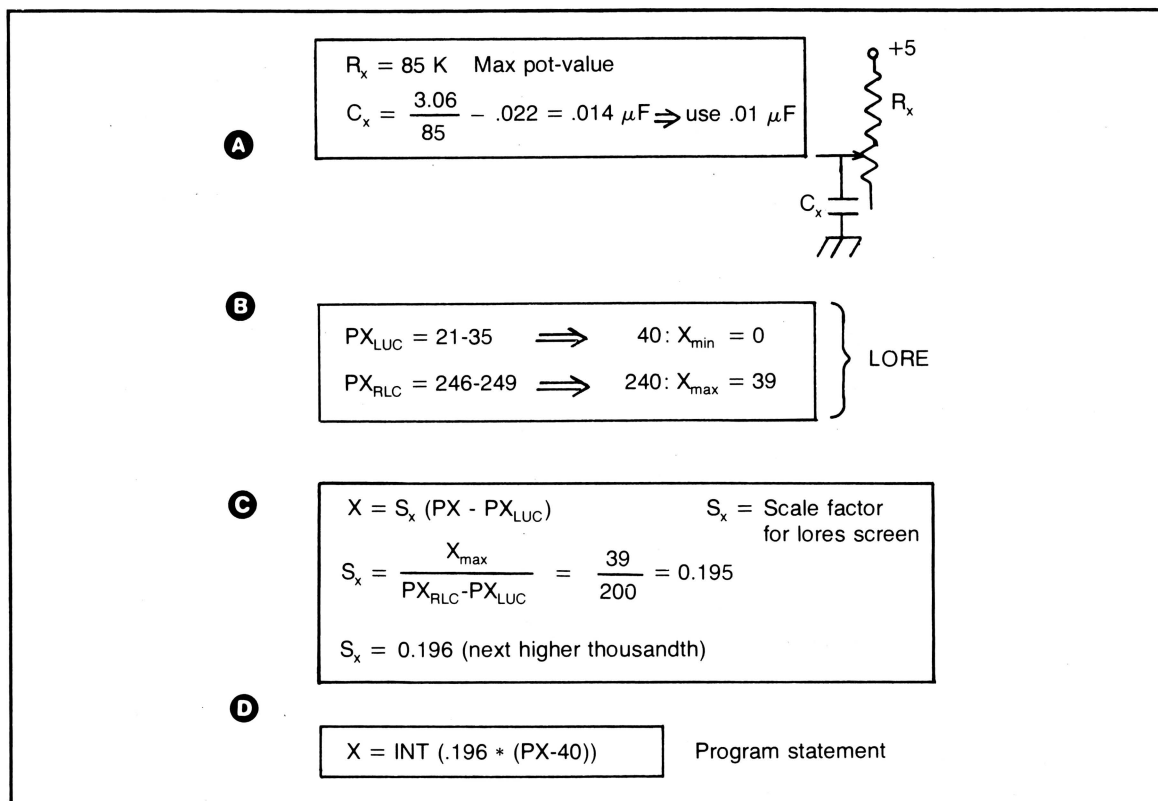
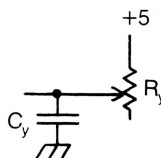


Fig. 11-6. Method for calculating scale factor for the X-pot, S_x .

A
 $R_y = 100 \text{ K max pot value}$

$$C_y = \frac{3.06}{100} - .022 = .0086 \mu\text{F} \Rightarrow \text{use } .005 \mu\text{F}$$

**B**

$$\begin{aligned} PY_{LUC} &= 28-32 \Rightarrow 40: & Y_{min} &= 0 \\ PY_{RLC} &= 245-247 = 240: & Y_{max} &= 39 \end{aligned}$$

C

$$\begin{aligned} Y &= S_y (PY - PY_{LUC}) \\ S_y &= 0.196 (PY - 40) \end{aligned}$$

$$S_y = \frac{Y_{max}}{PY_{RLC} - PY_{LUC}}$$

D

$$Y = \text{INT} (.196 * (PY - 40))$$

Program statement

Fig. 11-17. Method for calculating scale factor S_y .

the equation already introduced. From the calculation, a value of $.014 \mu\text{F}$ was obtained. For simplicity, we'll use a $.01 \mu\text{F}$ cap. and just add $.001 \mu\text{F}$ at a time, if necessary. So, for starters, let $C_x = .01 \mu\text{F}$.

Now do the same thing for the Y-pot, as given in Fig. 11-17A. In the real case, this pot actually turned out to have a value of 100 K. C_y was calculated at $.0086 \mu\text{F}$. Again, use a small value—.005 μF —and add in .001 increments if needed.

2. Using Listing 11-2 find out just what values are returned from PREAD when the joystick is moved between the two critical points—the left upper and right lower corners of the screen (LUC and RLC). You will have to find the two corners by reading the values of P_x and P_y as follows: the point on the joystick where the two values are at a minimum corresponds to the LUC of the screen; the point on the joystick where the two values are at maximum corresponds to the RLC of the screen. Mark these two points (LUP and RLP) on the joystick housing, perhaps temporarily, with tape.

Repeatedly take measurements at these two corners, and determine a range of values for both

PXmin and PYmin. For this example, real values of 21-35 and 28-32 were obtained for the PXmin and PYmin at the upper point of joystick excursion.

```

96 REM -----
97 REM  LORES. JOY. TEST1
98 REM -----
99 REM
100 HOME
110 PX = PDL (0):PY = PDL
    (1)
120 PRINT PX,PY
130 GOTO 110

```

Listing 11-2. This short routine gives PX and PY values at various points in the joystick excursion, before any screen display is created. This allows for proper conversion to screen (LORES/HIRES) coordinates.

These values roughly correspond to LORES graphics X,Y screen coordinates of (0,0).

Likewise, the PXmax and PYmax values obtained at the lower right point of the joystick excursion. Values in the range of 246-249 and 245-247 were obtained. These values correspond to the Xmax, Ymax LORES coordinates of (39,39). As a point of practical interest, the C_x and C_y values of .01 μF and .005 μF were all that were used to obtain the above P_x and P_y values.

From these PX and PY max. and min. values, 40 was chosen as a minimum value for both PX and PY, and 240 as a maximum value for PX and PY. It is these two values which will be used to scale PX and PY pairs to the proper screen coordinates.

Why do you get such variability at either end of joystick excursion? The most likely answer is the mechanical play inherent in cheap joysticks.

3. The formula used to calculate the X and Y coordinates

is based on the old $Y=mX+b$ type formula from high school math. As indicated in Fig. 11-16C and Fig. 11-17C, the scale factors S_x and S_y are calculated using the coordinate maxima on the LORES screen and the range between the PX and PY values at the LUP and RLP of joystick excursion. These were chosen as 40 and 240 in Step B.

A little calculation gives a value of 0.195 for both X and Y conversion. Round this value up to the next thousandth, 0.196. (this prevents roundoff error when the value of X and Y are calculated in BASIC).

4. Write the expression using BASIC statements.

5. Write a simple program which displays the values of PX and PY and also plot the joystick position on the LORES screen. The answer is given in Listing 11-3.

The program required some conditional

```
96 REM -----
97 REM LORES.JOY.TEST2
98 REM -----
99 REM
100 HOME : GR
110 PX = PDL (0):PY = PDL (1)
120 X = INT (.196 * (PX - 40)):Y
    = INT (.196 * (PY - 40))
130 IF X < 0 THEN X = 0
140 IF X > 39 THEN X = 39
150 IF Y < 0 THEN Y = 0
160 IF Y > 39 THEN Y = 39
170 COLOR= 0: PLOT XT,YT
180 COLOR= 1: PLOT X,Y
190 XT = X:YT = Y
200 PRINT "PX/X: ";PX;"/";X,"PY/
Y: ";PY;"/";Y
210 GOTO 110
```

Listing 11-3. Short program to illustrate the minimum statements needed to run the joystick presented here.

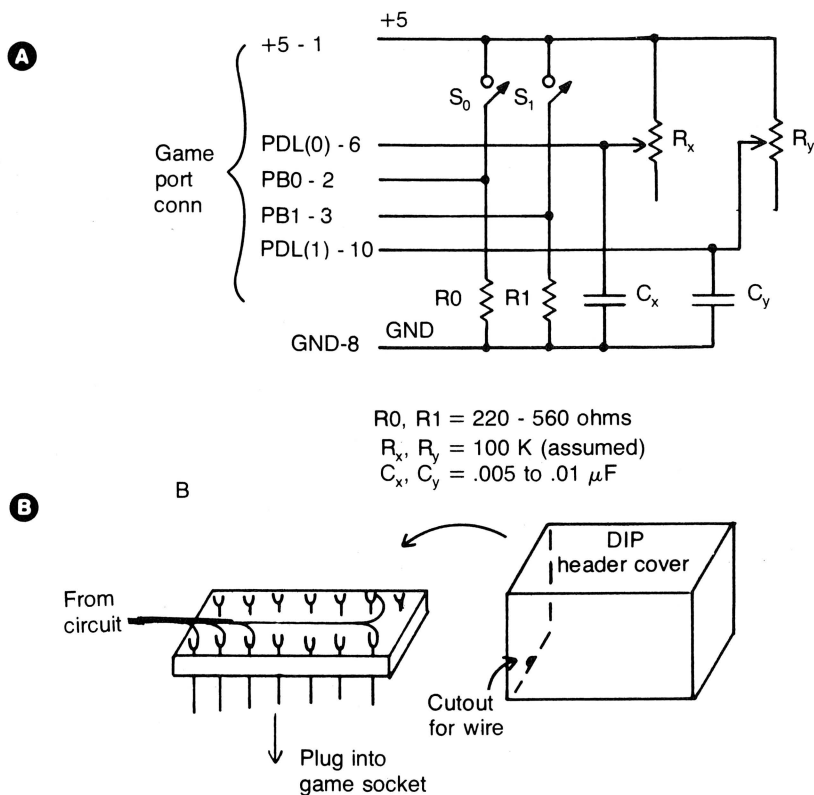


Fig. 11-18. A complete joystick + 2-pushbutton circuit.

statement to make sure that those points with coordinate values greater than 39 or less than zero were not plotted. These points lie within the shaded area of dead space in Fig. 11-15. Translating from these areas result in "illegal" coordinate values, which result in range errors. Therefore, the multiple IF statements in the listing are necessary.

A Complete Joystick Project

If you want to construct your own joystick, with two pushbuttons, use the circuit in Fig. 11-18 as a guide. The circuit can be housed in a small experimenter's plastic box. The buttons should be of the momentary pushbutton variety. Resistors R_0 and R_1 are pull-down, current limiting resistors for

the pushbutton inputs. The cabling can be six-conductor ribbon cable, commercial six-conductor round cable, or "homebrew" cable made of 22 gauge solid wire braided together.

Connect the wire to a DIP header. This is depicted in Fig. 11-18B. The six wires are soldered to their respective pins, and a small cutout in the DIP header cover is made to pass the wire, as indicated. The cover can be glued in place, and the whole assembly can then be plugged into the game socket. For a little labor and perhaps \$10 in parts you've got a servicable joystick.

HIRES Scaling

Try scaling the joystick to the HIRES mixed

(160 × 280) points. Hint: You have to make either one of two compromises—limited excursion on either end of the X-axis, or “dead points” along the X axis which cannot be reached. The latter problem will result if you try to stretch or scale a 200 point PX range into a 280 point HIRES X-axis range. See which compromise you find less objectionable.

Photocell Example

As a concluding project suggestion, you might try interfacing a resistive photocell (not a photo-voltaic). A Cadmium Sulfide photocell will change its resistance in response to a range of light intensities. Any such photocell will do—the Radio Shack 276-116A is an example, though any other over-the-counter or mail order source would do.

The typical resistive photocell has a characteristic dark resistance, usually on the order of 1 Megohm or so. Minimum resistance in very bright light may fall near or below 100 ohms. This represents a 10,000 to 1 ratio—about 40 times the range of the PREAD. The problem you should resolve is whether you want your setup to read very bright or very dim light intensities, or somewhere in between. As a rough guide, follow the procedure below:

Use an ohmmeter to check for the range of resistance in the illumination range you wish to measure. For a real-world case, I noted that the cadmium photocell approached 100 ohms in daylight, several thousand ohms in subdued room light. This seemed to be a reasonable range of resistance to work with.

With the photocell set up for the bright end of illumination, then, it is best to figure a minimum value for the time constant, based on the minimum value of its resistance in the brightest light you would expect to measure. (This is obviously different from the game pot example, because the whole intention of this transducer is different).

Referring to Fig. 11-19, calculate C2 on the basis of

1. The minimum PREAD time of 13 μ sec.
2. The minimum value of in-circuit resistance, 100 ohms.

The result for C2 would be about .11 μ F. Use a .10 μ F disc capacitor, and add additional .01 caps as necessary. Listing 11-1 is the minimum software you need to observe the variations in paddle value in relation to light intensity.

Is there some way of building a circuit to measure wide range of light intensity? Hint: use a switch arrangement of capacitors. A capacitor substitution box, if available, is ideal. What about other physical quantities? Thermistors, resistive strain gauges, pendula to measure mechanical oscillation (based on a common potentiometer) are some possibilities. If you use pots for angular measurement as in pendulum, make sure you use linear taper, not audio taper.

Another idea is to measure sound! Hint: carbon microphones act as resistive sound transducers.

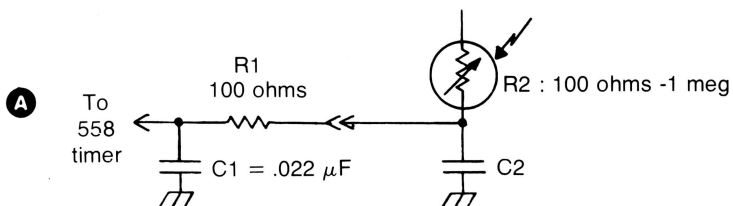
CLOSING COMMENTS

Hopefully, this will only be the beginning for you as a digital hardware experimenter. If you've mastered most of the material presented, and performed the experiments, then you have gained valuable skills as well as fundamental knowledge. There are numerous books which detail intermediate and advanced digital applications both in the form of stand-alone and computer-based circuits. Remember that manufacturers' application manuals are excellent sources of basic information and project ideas.

D/A conversion is an exciting area which you may want to look into at some point in your experimentation. Two dramatic examples are sound synthesis (covering data words into frequency/intensity information), and motor control (converting data words into mechanical motion). Sound synthesizer chips and stepper-motor controller chips are available for these more advanced projects.

An area that you might want to investigate is A/D conversion using dedicated chips—8 and 12 bit chips are cheap and readily available, but do require slot-based I/O hardware.

Another subject of interest is that of amplifica-



B

Min PREAD loop time = $13 \mu s$

$$TC_{\min} = R_{\min} (C1 + C2) = 13 \mu s$$

$$C2 = \frac{TC_{\min}}{R_{\min}} - C1 = \frac{13 \mu s}{100 \text{ ohm}} - .022 \mu F$$

$$C2 \cong 0.11 \mu F$$

Fig. 11-19. Method for calculating C2 for a low resistance range in a cadmium sulfide photocell.

tion, necessary for measuring very weak signals. Therefore, a knowledge of operational amplifiers (integrated amplifiers on a chip, which are linear

devices) is certainly important in some of the more advanced A/D applications, when you are ready to explore them.

Index

Index

A

Absorption, 67, 68
Accumulators, 298
Acknowledge signal, 303
Active high, 90
Active low, 90
Adders, 297
Address decoding, 185, 303
Address signals, 181
Address word, 10
Ampere, 94
Amplification, 144
Analog, 8
Analog data, 303
Analog inputs, 320
AND, 48
AND-form, 48, 64
Annunciator outputs, 15
Apple computer, 2
Apple Reference Manual, 16
Applesoft, 2
Arithmetic logic units, 298
Arsenic, 124
ASCII, 275
Associative Law, 68
Astable devices, 190
Asynchronous, 205
Asynchronous circuits, 197

B

Base, 138
Base-two, 8
BASIC, 181
BCD, 240, 269
BDIS, 2, 20, 21
BDIS, entering, 26
BDIS program listing, 22-25
Beta, 144
Bias, transistor, 101
Binary, 8
Binary coded decimal, 240
Bipolar transistor, 4
Bistable devices, 190
Bit, 8
Bit trains, 259
Black box, 2, 3, 161, 163, 164
Blanking, 282
Block decoding, 310
Boolean Algebra, 61
Boolean Algebra, laws of, 67
Boolean equations, 74
Boolean expression, 61
Boolean operators, 61
Boolean postulates, 67
Boolean rules, 65
Breadboard, 2
Breadboard strip, 2, 14

Breadboard-In-Software, 2
Buffer, 185, 303
Buffer-drivers, 185
Bus, address, 181
Bus, data, 181
Bus, shared, 182
Bus, system, 300
Bus isolation, 303
Buses, 181
Byte, 10, 181

C

Capacitance, 103
Capacitor, computer grade, 116
Capacitor, disk, 116
Capacitor, electrolytic, 116
Capacitor component law, 109
Capacitor ratings, 115
Capacitor values, 115
Capacitors, dc blocking, 117
Capacitors, despiking, 118
Capacitors, power supply filter, 116
Cascaded, 274
Cassette input, 14, 17, 318
Characteristic curve, 127, 130-132
Charge, 92
Charge distribution, 124
Charging current, 110

- Chip select, 303
- Clipper circuit, 134
- Clock, 196
- Clocked logic, 195-197
- Clocks, 190
- Code conversion, 269
- Collector, 138
- Combinational devices, 37-60
- Combinational devices, MSI, 237, 269-298
- Combination devices, 189
- Common emitter, 141
- Commutative Law, 68
- Comparators, 291
- Computer, Apple, 2
- Computer logic, 181
- Control bus, 304
- Control signals, 181
- Corer circuit, 134
- Coulomb, 93
- Counters, 197
- CPU, 181
- Current, 92, 93, 97
- Current, charging, 110
- Current, leakage, 110
- Current gain, 144
- Current sink, 147-150
- Current source, 150-152
- Current transients, 118

D

- Data bus, 276
- Data distributor, 272
- Data latch, 303
- Data routing, 269, 276
- Data selectors, 271
- Data signals, 181
- Debouncer, switch, 193
- Decay, 107
- Decimal number, 8, 10
- Decoder, 269, 272, 276
- Decoder, seven-segment, 270, 281
- Decoding, I/O address, 310
- Decoding circuitry, I/O, 312
- Delay line, 175, 228
- De Morgans Theorem, 67, 70
- Demultiplexer, 276
- Demultiplexers, 271
- Desaturation, 152
- Despiking, 181
- Device count, 81
- Device family, 2, 4
- Device select, 303
- Device select chip, 315
- D flip-flops, 201, 202
- Dielectric, 109
- Digital data, 303
- Diode, 127
- Diode, zener, 133
- Diode logic, 136
- DIP jumper, 2, 14

- Disable, 183
- Discharge, 107
- Discrete components, 2, 3, 91
- Distributive Law, 68
- Divide by 2, 259
- Doping, 124
- Dual rail outputs, 190

E

- Edge-triggered, 196-198
- Electronics, digital, 1
- Electrons, 123
- Emitter, 138
- Enable, 183
- Enabling, 48
- Encoder, 272
- Encoder, keyboard, 275
- Encoder, priority, 272
- Encoders, 271
- Equations, Boolean, 74
- Equations, schematics from, 77
- Equipment, additional, 27
- Equipment, necessary, 2, 18, 19
- Exclusive-OR, 291-297
- Experiment 1, Setup, 31
- Experiment 2, NOT and IS, 44
- Experiment 3, AND, 48
- Experiment 4, NAND, 51
- Experiment 5, OR, 56
- Experiment 6, NOR, 57
- Experiment 7, SSI Design I, 84
- Experiment 8, SSI Design II, 87
- Experiment 9A, diode and transistor testing, 156
- Experiment 9B, transistor current gain and the overloaded sink, 158
- Experiment 10, LSTTL sink current measurement and augmentation, 185
- Experiment 11, R/S latches, 198
- Experiment 12, the LS74 flip-flop, 206
- Experiment 13, the Ls75 quad D-latch, 209
- Experiment 14, the JK flip-flop, 214
- Experiment 15, the one shot, 231
- Experiment 16, MSI ripple counter, 241
- Experiment 17, the synchronus Up/Down counter, 249
- Experiment 18, the shift register, 260
- Experiment 19, priority encoder, 284
- Experiment 20, binary decoder/demultiplexer, 285
- Experiment 21, BCD encoder, 287
- Experiment 22, seven-segment decoder, 287
- Experiments, 31
- Experiments, design, 84
- Experiments for combinational MSI, 283
- Exponential, 104

F

- Family of devices, 4
- Fan-out, 170
- FET, 4
- Firmware, 305
- Firmware, system, 306
- Flip-flop, 190-223
- Flip-flop, JK, 212
- Form, 64
- Forward bias, 129, 130, 139
- Functional description, 40

G

- Gallium, 124
- Game port, 2, 13
- Gate, 6, 194
- Gate equivalence, 37, 48
- Gating, 48
- Ground, 18
- Grounding, 180

H

- Half adder, 297
- Hamming code, 297
- Handshaking, 302, 303
- Hexadecimal, 11
- Hex inverter, 40
- Holes, 125
- Hysteresis, 222

I

- IC, digital, 1
- IC, internal makeup of, 161
- Inputs, unused, 180
- Integration, scale of, 5-8
- Interface circuitry, 300
- Interfacing, 299
- Internal capacitance, 103, 127, 132
- Interrupt handler, 276
- Interrupt servicing, 276
- I/O select chip, 312
- I/O triad, 299
- IS, 44

J

- JK flip-flop, 212
- Joystick interfacing, 325
- Joystick project, 329-333
- Junction barrier voltage, 127-130

K

- Keyboard, 275
- Kirchhoff's current law, 97
- Kirchhoff's laws, 91
- Kirchhoff's voltage law, 96

L

- Latch, 190, 193
- Latch, addressable, 278
- Latch, octal, 262
- Latch, transparent, 205

LCD, 270, 283
 Leading edge, 174, 193
 Leakage current, 110
 LED, 270, 281, 283
 LEDs, 136
 Level-triggered, 198
 Light emitting diodes, 136
 Loading, 152
 Loading rules, 180
 Logic, clocked, 195
 Logic, synchronous, 196
 Logic high, 8
 Logic low, 8
 Logic trainer, 2, 18
 LS00, 202
 LS04, 40
 LSI, 6, 189
 LS14, 222
 LS47, 283
 LS48, 281
 LS49, 283
 LS74, 205
 LS75, 205
 LS76, 233
 LS86, 291, 295
 LS93, 234
 LS123, 223, 231
 LS125, 185
 LS126, 185
 LS138, 276
 LS148, 275
 LS181, 298
 LS190, 248
 LS192, 248
 LS193, 246
 LS194, 255
 LS197, 237
 LS240, 289, 290
 LS241, 289, 290
 LS245, 290
 LS259, 280
 LS280, 298
 LS281, 298
 LS283, 298
 LS371, 236

M

Majority charge carrier, 124
 Master-slave principle, 214
 Maxterms, 64
 Mean time before failure, 234
 Memory, 189
 Memory, on-board I/O, 308
 Memory, volatile, 305
 Memory blocks, 304, 305
 Memory map, 304
 Memory map, I/O, 304, 308
 Memory map, system, 304, 305
 Memory mapped I/O, 301
 Memory mapping, 301
 Memory pages, 304, 305

Minority charge carrier, 124
 Minterms, 64
 Mixed gate implementations, 81
 Monostable devices, 190, 219
 MOS, 4
 MSI, 6, 233
 MSI functions, 235
 Multiplexer, 276
 Multiplexers, 271
 Multiply by 2, 259
 Multivibrators, 190

N

NAND, 51
 NAND/NAND logic, 80
 Natural base, 107
 Nibble, 10
 Noise, 174, 177
 Noise immunity, 170, 172, 177
 Nomenclature, device, 39
 NOR, 57
 NOR/NOR logic, 80
 NOT, 44
 N-type material, 124

O

Octal latch, 262
 Ohm, 95
 Ohm's law, 91, 94
 On-board I/O chip, 315
 One shot devices, 223
 One shots, 190, 219
 Open collector operation, 164
 Operational amps, 226
 Operators, Boolean, 61
 OR, 56
 OR-form, 48, 64
 Overhead, 81, 233, 234
 Overloading, 152

P

Package count, 81
 Paddle inputs, 320
 Page boundary, 305
 Parallel circuits, 97
 Parameters, electrical, 3
 Parameters, TTL, 163, 170
 Parity, 295
 Parity, even, 295
 Parity, odd, 295
 Parity checker, 295
 Parity generator, 295
 Peak inverse voltage, 132
 Peripheral data format, 302
 Peripheral device, 300
 Peripheral signal conditioning, 302
 Photocell project, 333
 Physical package, 38
 Place value, 8
 Polarity, 96, 128
 Postulates, Boolean, 67

Power, 18
 Power consumption, 174, 177
 Power consumption, estimates of, 179
 Power spikes, 118
 Precautions, 31
 Precautions, handling, 42
 Printer, 26
 Priority rules, 62
 Product of sums form, 64
 PROM, 301, 371
 Propagation delay, 175, 193
 Propagation time, 174
 P-type material, 124
 Pull-down, 43, 180
 Pull-up, 43, 180
 Pulsating dc, 133
 Pulse-width, 174, 227
 Pushbutton inputs, 16, 320

R

RAM, 301
 RAM memory, 190
 Reactance, 103, 113
 Readkey, entering, 26
 Readkey listing for monitor entry, 27
 Readkey miniassembler listing, 27
 Readkey program listing, 25, 26
 Ready signal, 303
 Refractory period, 219
 Reliability, 234
 Resistive circuits, 101
 Resistive transducers, 229
 Resistor, 91
 Resistor, ac, 114
 Resistor ratings, 99, 100
 Resistor tolerance, 99
 Resistor types, 99
 Resistor values, 99
 Reverse bias, 130, 131
 Ripple counters, 218, 237-249
 ROM, 301
 ROM select chip, 312
 R/S flip-flops, 190-201

S

Saturation, 142
 Scale of integration, 5, 189
 Schematics from equations, 77
 Schmitt trigger, 220, 221-223
 Schottky, 5
 Schottky diode, 146
 Schottky transistor, 5
 Semiconductor, 123
 Sequence generators, 259
 Sequential devices, 38
 Sequential devices, LSI, 189-233
 Sequential devices, MSI, 233-267
 Sequential MSI logic, 235
 Series, 147
 Series circuits, 96
 Set-up time, 176, 177

- Shift registers, 255
- Shift right operation, 255
- Shunt, 99, 147
- Silicon atom, 123
- Silicon crystal, 124
- Simplification, 72
- Sink, 147, 162
- Softswitches, 16
- Software, 229, 275, 301
- Source, 147, 162
- Speed, 174
- SSI, 6
- SSI design experiments, 84-90
- State table, 193
- Structured design, 234
- Substrate, 123
- Sum of products form, 64
- Switch debouncer, 193
- Switching noise, 177
- Synchronous, 205
- Synchronous counter, 241
- Synchronous logic, 196
- System bus, 300

T

- T flip-flops, 201, 202

- Terms, 61, 64
- Three-state devices, 289
- Three-state logic, 182
- Time constant, RC 119-121
- Time constants, 103-114
- Time derivative, 94, 113
- Timer, 558, 226
- Timers, 190
- Timing diagram, 192
- Totem pole operation, 167
- Trailing edge, 174
- Transconductance, 142
- Transfer curves, 142
- Transfer resistor, 123, 140
- Transistor, 4, 123
- Transistor, bipolar, 138
- Transistor action, 138
- Transistor bias, 101
- Transistor parameters, 138
- Transistor ratings, 155
- Transistor switch, 147
- Trigger, 190
- Truth tables, 40
- Truth tables, equations from, 74
- TTL, 4

- TTL Data Manual, 30
- Twisted pairs, 258

U

- ULSI, 7
- Unit pulses, 114

V

- Valence shell, 123
- Variables, 61
- VLSI, 6
- Volatile memory, 305
- Voltage, 92, 97
- Voltage clamp, series, 134
- Voltage clamp, shunt, 134
- Voltage division, 96
- Voltage drop, 96
- Voltage drop, polarity of, 96
- Voltmeter, 27, 28

W

- Watt, 101
- Word, 10

Interfacing and Digital Experiments with Your Apple®

by Charles J. Engelscher

- Use your Apple microcomputer as a desk top laboratory for exploring a fascinating new world of digital technology and I/O applications!
- Master the principles of digital electronics from gate level to MSI devices!
- Get a working knowledge of Apple I/O organization, essential interfacing principles, and simple A/D methods!

It's easy . . . it's expensive . . . it's a totally unique new approach to learning and experimenting with digital hardware. The secret: a structured software program called Breadboard-in-Software (BDIS) that allows you to learn digital techniques by hands-on experimentation. No expensive new equipment or modifications to your standard Apple are needed. In fact, all you need is an inexpensive dip-jumper and a solderless socket strip, your Apple, *and* this book to get started.

You'll find out how to conduct a wide range of experiments using the Apple game port to turn your screen into a dynamic, reconfigurable truth table. You'll learn about fundamental digital concepts like NAND and NOR logic, Boolean methods, TTL internals . . . SSI packages, sequential logic, timer ICs, and design methods . . . MSI functions, I/O concepts, and A/D conversions.

If you're interested in learning more about digital logic and interfacing, you'll find no more practical or inexpensive method than this book provides!

Charles J. Engelscher holds a B.S. in Electrical Engineering and is also an M.D. whose specialty is diagnostic radiology. He is currently Director of the Department of Radiology for a hospital in upstate New York.

TAB TAB BOOKS Inc.

Blue Ridge Summit, Pa. 17214

Send for FREE TAB Catalog describing over 750 current titles in print.

FPT >

PRICES HIGH

ISBN 0-8306-1717-5

1495-0684

117



ENGELSHERR

INTERACING & DIGITAL EXPONENTS
WITH YOUR APPRE[®]